ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

PREGUNTAS Y RESPUESTAS

Diego Sánchez Díaz

10 de junio de 2010

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS

PREGUNTAS Y RESPUESTAS

- Departamento: Lenguajes y Sistemas Informáticos
- Directores del proyecto: Manuel Palomo Duarte y Juan Boubeta Puig
- Autor del proyecto: Diego Sánchez Díaz

Cádiz, 10 de junio de 2010

Fdo: Diego Sánchez Díaz

Agradecimientos

Me gustaría agradecer a mi familia: mi madre, mi padre, mi hermana y a Elena, por el tiempo que me han dedicado a ayudarme a probar el juego las veces que he necesitado y por los consejos recibidos por ellos.

Licencia

Convenciones, notación y formato

Para facilitar la lectura y comprensión del documento se citan las principales convenciones, notaciones y formatos utilizados durante la escritura de dicha memoria:

- Los comandos que han de usarse en la consola de cualquier sistema GNU/Linux tendrán el formato:

```
Esto es una prueba de comandos en una consola
```

- Cuando hablemos de los concursos que componen el software cumplirá: **Normal**, **Contrarreloj**.
- Cuando hablemos de programas y herramientas usadas durante el desarrollo del proyecto usaremos el formato: *GIMP*, *Planner*.

Índice general

Indice de figuras

Capítulo 1

Introducción

1.1. Objetivos

El objetivo de este Proyecto Fin de Carrera (PFC) es crear un videojuego de preguntas y respuestas. Se desea crear un videojuego intuitivo, fácil de usar y con una interfaz agradable para que resulte atractivo para los jugadores. Un videojuego con preguntas personalizadas por los usuarios, para poder jugar en el contexto de cada grupo. Este puede ser utilizado por estudiantes para realizar sus propios paquetes de preguntas de diferentes asignaturas y afianzar conceptos de una forma divertida.

Con este tipo de videojuego, se pretende que exista un juego de software libre en el que los usuarios pueden crear sus propias preguntas de una forma fácil e intuitiva y sin necesidad de tener grandes conocimientos informáticos. Esto hace que el juego sea extensible, ya que se pueden cambiar o añadir tantas preguntas como el usuario considere oportuno, así como la temática de las mismas.

1.2. Glosario

1.2.1. Acrónimos

GIFT: Formato estándar de preguntas utilizado por Moodle.

GFDL: GNU Free Documentation License.

LGPL: GNU Lesser General Public License.

GPL: General Public License.

GNU: GNU is Not Unix.

SDL: Simple DirectMedia Layer.

OSLUCA: Oficina de Software Libre y Conocimiento Abierto de la Universidad de Cadiz.

CPU: Central Processing Unit.

BMP: Beep Media Player.

JPEG: Joint Photographic Experts Group.

TIFF: Tagged Image File Format.

PNG: Portable Network Graphics.

PNM: Portable aNy Map.

PCX: PiCture eXchange.

XPM: X PixMap.

GIF: Graphics Interchange Format.

TGA: Truevision TarGA.

Wave: WAVeform Audio File Format.

MP3: MPEG-1 Audio Layer 3.

OGG: Formato para música.

MOD: Formato de sonido parecido a midi.

S3M: Formato para música.

IT: Formato para música.

XM: Extended Module.

TTF: True Type Font.

1.3. Tecnologías utilizadas

En esta sección se presentan las herramientas que he utilizado para el desarrollo del juego.

- C++: Lenguaje de programación orientado a objetos. Todo el videojuego se ha realizado con este lenguaje.
- *libSDL*: Librería gráfica basada en C/C++. Es la base gráfica de mi videojuego. Es libre y la componen varios subsistemas: video, sonido, eventos, etc.
- GIFT: Este formato estándar de preguntas ha sido el escogido para representar las preguntas de una forma fácil e intuitiva. Nos dá una estructura para escribir preguntas fácil de crear y de leer.
- *GIMP*: Es una herramienta de dibujo muy completa, simple y libre. La he utilizado para la realización de fondo, botones y todo tipo de imagenes, así como la modificación de imagenes libres.

- *Sinfyg*: *Synfig* es una poderosa herramienta de animación y diseño 2D vectorial, programa open-source creado por Robert Quattlebaum con la ayuda adicional de Adrian Bentley, diseñado para producir películas animadas con pocas personas y recursos.
- *Planner*: Herramienta para el desarrollo de diagramas de Gantt y de tareas.

1.4. Estructura del documento

El documento se divide en nueve capítulos, el primero de ellos es en el que nos encontramos.

En el segundo capítulo se encarga de comentar los conceptos básicos del proyecto, así como hablar del tema del que trata el videojuego y las tecnologías que se han utilizado.

En el tercer capítulo se explica las tareas realizadas y el tiempo que se ha dedicado a cada una de ellas , ya sean tareas gráficas como tareas de implementación de código. Todo ello se presentará mediante un Diagrama de Gantt[1] usando para ello la herramienta libre *Planner* incluida en los repositorios de cualquier distribución GNU/Linux.

En el cuarto capítulo se comenta la fase de análisis que se ha realizado para finalizar el producto. Se incluyen algunos diagramas entidad-relación, más conocidos como ERs.

En el quinto capítulo se expone la fase de diseño del videojuego, es decir, definir el sistema con total detalle.

En el sexto capítulo se habla sobre la implementación del videojuego, las técnicas y herramientas usadas, así como las diferentes estrategias que se han seguido para realizarlo.

En el séptimo capítulo se muestran las diferentes pruebas a las que ha sido sometido el producto.

En el octavo, expongo las conclusiones a las que he llegado después de la finalización del proyecto, lo que he aprendido, posibles extensiones del proyecto, etc.

En el último capítulo se incluye distintos apéndices tales como el manual de usuario, manual de instalación, manual para agregar nuevas preguntas y categorías, biblioteca de preguntas en formato GIFT, etc.

Al ser un producto de Software Libre se incluye la licencia GFDL 1.3 en la que se basa la documentación del proyecto.

[1] El Diagrama de Gantt es una popular herramienta cuyo objetivo es mostrar el tiempo de dedicacación previsto para diferentes tareas o actividades a lo largo de un tiempo determinado.

Capítulo 2

Conceptos Básicos

2.1. Visión Global del Producto

El producto que se presenta responde a la demanda de crear un juego de preguntas y respuestas con licencia GPL y capaz de expandirse con nuevas preguntas y temáticas.

Existe un proyecto similar y también distribuido bajo licencia GPL e integrado en el proyecto GNU, Quizzer [1], pero está abandonado.

El producto software resultante de este proyecto, Prerres, está pensado para ser un juego que sea totalmente extensible. Un producto en el que cualquier usuario sea capaz de añadir nuevas preguntas o modificar las ya existentes con nuevas respuestas. Uno de los grandes problemas de los juegos privativos, como puede ser el caso del videojuego de PlayStation®, Buzz!™, es que una vez que el jugador ha jugado varias veces, las preguntas se repiten y es fácil que llegue un momento en el que se saben las respuestas a todas las preguntas. Entonces, el jugador tiende a perder la atracción y el interés de seguir jugando. Además, si se quiere añadir una nueva colección de preguntas se deben comprar como pack en PlayStation®3 o adquirir otro juego de Buzz!™, para PlayStation®2. Por no hablar de que no se pueden personalizar el tipo de preguntas que se hacen en el concurso o si se puede, es de una forma muy limitada. Posiblemente estos sean los mayores problemas que tienen cualquier videojuego en la actualidad que posea el carácter privativo.

Imaginemos un videojuego libre y gratuito, en constante expansión, en el que cada vez que el usuario se canse de jugar, pueda cambiar completamente el tipo, número o categoría de las preguntas y respuestas de las que dispone el juego. Esto es a grandes rasgo Prerres.

Con esto, no hay que depender de que el creador del videojuego acceda a introducir los cambios que el usuario le exponga, sino que el propio usuario es el que puede personalizar su juego.

En la idea anteriormente descrita, está la originalidad del proyecto Preguntas y Respuestas, crear un videojuego totalmente personalizable para el usuario sin conocimientos en programación. Además, si el usuario dispone de conocimientos de programación, es posible añadir nuevos tipos concursos de forma sencilla.

Crear juegos libres en constante crecimiento, sería una opción que proporcionaría protagonismo a los juegos bajo licencia libre frente a los juegos privativos.

2.2. Antecedentes

La idea de este tipo de juegos no es nueva, a continuación vamos a repasar cómo apareció el tipo de juego de preguntas y respuesta.

El primero en sacar un tipo de juego de este estilo fue PlayStation®, con la serie Buzz!™. A continuación se indica de forma cronológica los diferentes títulos que componen esta serie:

- **Buzz!: El Gran Concurso Musical** para PlayStation®2, Octubre de 2005. Un juego de preguntas y respuestas sobre los últimos 50 años de música Pop en el que no falta detalle, ni concursantes rivales, ni presentador enrollado.
- **Buzz!: El Gran Reto** para PlayStation®2, Abril de 2006. De nuevo el mismo sistema con preguntas de cine, deportes y cultura general y la posibilidad de ocho jugadores simultáneos.
- **Buzz!: El Gran Concurso de Deportes** para PlayStation®2, Noviembre de 2006. Esta vez con todo tipo de preguntas referentes a fútbol, baloncesto, tenis, golf, etc.
- **Buzz!: El Mega Concurso** para PlayStation®2, Mayo de 2007. Este título añade nuevas rondas y modos de juego, hasta un total de 5.000 nuevas preguntas de conocimiento general, televisión, películas, música, deportes, ciencia, naturaleza y mucho más.
- **Buzz!: The Hollywood Quiz** para PlayStation®2, Noviembre de 2007.En este título se incluyen preguntas sobre cine.
- **Buzz!: El Multiconcurso** para PlayStation®3, Julio de 2008. Un nuevo concurso de la serie Buzz, ahora para PS3 y permitiendo por primera vez generar nuestras propias preguntas, aunque de forma bastante pobre.

La serie dió un enfoque distinto a la saga con juegos pensados para los más pequeños. A continuación se indica de forma cronológica los diferentes títulos para los más pequeños:

- **Buzz! Junior: Locura en la Jungla** , Octubre de 2006.
- **Buzz! Junior: Robo Jam**, Mayo de 2007.
- **Buzz! Junior: Monster Rumble**, Noviembre de 2007.
- **Buzz! Dinomania**, Octubre de 2008.

Buzz! Junior se caracteriza por desarrollar entre los jugadores habilidades de atención, rapidez, agilidad, instinto, lógica . . . perfectamente desarrolladas mediante un envoltorio más que apetecible. Jugar, aprender y desarrollar los reflejos son algunas de las aclamadas características de estos juegos basados en el uso de los llamados Buzzers. Esta serie todavía no ha dado el paso a PlayStation®3.

Hace poco, en Marzo de 2009, la compañia Electronic Arts lanzó al mercado el juego de mesa de toda la vida, Trivial Pursuit, en formato digital para la consola Wii™. En está nueva versión se han añadido nuevas funcionalidades con respecto a la versión de mesa. Nuevos modos de respuesta, distintos modos de juego y ha conseguido sacar el mejor partido a los mandos de Wii™.

2.3. Alcance

Se pretende que el videojuego tenga la dificultad que el usuario quiera darle. Con la finalidad de que el usuario se implique en el juego como algo suyo, ya que básicamente él va a ser el que introduzca preguntas y respuestas de la dificultad que él crea conveniente. Esto lo hace totalmente diferente a los videojuegos mencionados anteriormente, en el que todo viene predeterminado por los desarrolladores (preguntas, respuestas, dificultad). Lo único que no puede modificar el usuario son los tipos de concursos, a menos que posea conocimientos en programación y más concretamente en C/C++. Además, al estar el código basado en C/C++ y libSDL, éste se hace más familiar para cualquier informático con conocimientos del lenguaje y de la librería anteriormente descrita.

Capítulo 3

Planificación

3.1. Incrementos

Debido a la naturaleza del proyecto, se ha decidido que lo más adecuado es utilizar un **modelo incremental**. Crear una primera iteración e ir añadiendo nuevas funcionalidades es la idea que más se adapta a este tipo de proyecto, ya que dispone de una base sólida como es la biblioteca GIFT y el gestor de menús, y lo demás es ir añadiendo funcionalidades que es lo que se va realizando en los distintos incrementos. Esto ayuda a no ver el videojuego como un todo, sino como pequeñas partes que se van incorporando. Además, al ser yo la única persona que realiza el proyecto, permite un desarrollo más eficiente del software.

Como sabemos,en el primer incremento sólo se consideran los requisitos básicos del sistema, en este caso la especificación inicial de requisitos, las imágenes, código de los menús, etc. Una vez realizado el primer incremento, se prueba y a partir de él se planifican las modificaciones a realizar en el siguiente incremento y así progresivamente hasta conseguir el producto final.

A continuación se muestran los principales incrementos realizados en el proyecto **Preguntas y Respuestas**.

3.1.1. Incremento 1: Requisitos básicos del sistema

En este incremento se ha planificado el tiempo que se dispone para la realización del proyecto y las distintas tareas que se deberían realizar en cada momento. El tiempo inicial de trabajo será de 4 de la tarde a 8 de la noche. Por tanto, se intentará realizar cada día o días una tarea distinta, con el fin de hacer más liviano el trabajo de realizar un proyecto largo.
Se inicia con la especificación de lo que va a realizar globalmente el juego, es decir, lo que se quiere conseguir con el videojuego sin pensar en los distintos concursos u opciones que lo componen.
Gracias al programa *GIMP*[1] se realizan las pantallas de juego, así como todas las imagenes o modificaciones de imagenes libres que se han necesitado en el videojuego.

[1] *GIMP* (GNU Image Manipulation Program) es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito. Está englobado en el proyecto GNU y disponible bajo la Licencia pública general de GNU (GPL)

Figura 3.1: Imagen de la pantalla principal de Prerres realizada mediante GIMP

Al no conocer LaTeX, asistí a un curso de LaTeXpara proyectos fin de carrera que impartió un compañero de la Universidad de Cádiz y organizado por la OSLUCA, y dispuse de las plantillas que he utilizado para realizar la memoria. Además, consultando el libro **LaTeX: una imprenta en sus manos**[2] no me resultó dificil redactarla. De esta forma desarrollé la primera versión de la memoria del proyecto.

3.1.2. Incremento 2: Organización de las preguntas

En este incremento, decido que estructura voy a utilizar para almacenar las preguntas en memoria primaria. Después de barajar diferentes opciones, me decanto por separar en dos clases este trabajo: la clase Pregunta, que será la que guarde una pregunta concreta y la clase Lote que será la que haga de biblioteca de preguntas e incluya el algoritmo para pasarlas de memoria secundaria a memoria principal.
Una vez decidido, paso a diseñar e implementar la clase Pregunta. Esta clase se expande en una clase por cada tipo de pregunta que admite el formato GIFT. Por tanto, tenemos las siguientes clases:

- Clase MultiOpcion.
- Clase MultiRespuesta.
- Clase Corta.
- Clase Numerica.
- Clase Relacion.
- Clase TrueFalse.

En el videojuego sólo vamos a utilizar la clase MultiOpción al ser las preguntas que más juego van a dar y las más fáciles de modificar para el usuario. Aún así, esto da lugar a una biblioteca de preguntas que se explica de forma más detallada en el Apéndice D.
Seguidamente, paso a diseñar e implementar la clase Lote. Está clase es la que va a utilizarse como biblioteca de preguntas, y también incluye el algoritmo para pasar las preguntas de memoria secundaria y memoria principal. Para ello hemos utilizado un script realizado por Casiano Rodríguez León, *Gift-0.6*[3], que genera un código de gramática más simple que la del formato gift. Esto ha simplificado el trabajo, ya que al ser una gramática sin ambigüedad, el esfuerzo para leer el fichero ha sido menor.

Este incremento necesitó una dura fase de pruebas para comprobar que el algoritmo funcionara correctamente en todos los casos. Es una de las partes más complicadas del proyecto y la que me ocupó más

tiempo de desarrollo.

3.1.3. Incremento 3: Diseño e Implementación de clases básicas

En este incremento me centro en diseñar e implementar las clases básicas del sistema:

- Clase Inicio: Inicializa la SDL.
- Clase Sonido: Controla el subsistema de sonido de la SDL.
- Clase Video: Controla el subsistema de video de la SDL.
- Clase Fuente: Permite escribir una frase en la pantalla.
- Clase Imagen: Permite cargar una imagen a una superficie de la SDL.

3.1.4. Incremento 4: Diseño e Implementación de la clase Juego

Una vez terminada la base del juego, diseño e implemento la clase principal del videojuego (Clase GestorMenu) en la que se llaman a los diferentes menús y gestores que se quieran ejecutar. Después me centro en darle funcionalidad a las distintas opciones que hay en el menú principal del videojuego. Implemento la opción de activar o desactivar el sonido, cambiar el idioma de la aplicación y actualizar las preguntas en tiempo de ejecución.

En este incremento se decide finalmente la utilización por defecto de mandos para contestar a las preguntas. En especial los buzzers[2] de PlayStation®2. Esto me hace tener que reimplementar el código del menú en una nueva clase (Clase Menú) y crear un nuevo estilo de elección.

3.1.5. Incremento 5: Planificación de concursos e implementación de clases

En este incremento se redactan los diferentes concursos que vamos a implementar en el videojuego, creando sus reglas, el sistema de puntuación, así como las clases que necesitamos. Para ello se han utilizado los criterios propios y los requeridos por los usuarios finales del videojuego.

Una vez analizado el sistema de concurso diseñamos e implementamos las siguientes clases:

- Clase GestorConcurso: Gestiona los distintos concursos.
- Clase Jugador: Representa un Jugador con su nombre, puntuación, personaje...
- Clase Cronometro: Representa un cronómetro.
- Clase Concurso: Esta clase es donde se van a implementar todos los concursos que vaya a crear.
- Clase Simple: Esta clase es una generalización de los concursos simples.
- Clase Contrarreloj: Esta clase sirve para controlar el concurso **Contrarreloj**.

En este incremento además se diseñan e implementan las opciones de los concursos, entre ellas se encuentran:

[2]Los buzzers son mandos especialmente diseñados para la serie Buzz!™

- Número de jugadores y elección de nombre.
- Elección de categoría de las preguntas.
- Número de preguntas (Unicamente para concursos simples).
- Tiempo de contrarreloj (Unicamente para concurso contrarreloj).

3.1.6. Incremento 6: Diseño e Implementación Concurso Normal

Este concurso es el más sencillo de todos.

Su funcionamiento es simple, al empezar el concurso, el turno es del Jugador 1, esté debe responder a una pregunta que se le presenta por pantalla en un tiempo determinado. En caso de que no conteste correctamente se le restan 50 puntos. En caso de acertar se le suma 100 o menos puntos, dependiendo del tiempo de respuesta. En cualquiera de los dos casos se pasa el turno al siguiente jugador que se le formula una pregunta distinta. Ganará el que más puntos consiga.

El concurso se modela mediante la clase Normal. Esta clase guarda toda la información necesaria para el seguimiento del concurso y lo ejecuta.

Una vez implementada, los usuarios se quejan de que no está muy claro cuando le toca el turno a ellos. Esto se soluciona dibujando de otro color, el recuadro del jugador, así como su número de jugador, puntuación y cronómetro.

3.1.7. Incremento 7: Diseño e Implementación Concurso Contrarreloj

En este incremento se ha añadido el concurso **Contrarreloj**. Este concurso también es bastante simple. Cada jugador dispone de un tiempo determinado en las opciones del concurso. Este tiempo irá descendiendo en su turno mientras no responda la pregunta que se le formula. Así para todos los jugadores. Deberán contestar tantas preguntas como tiempo les quede. El concurso finalizará cuando a todos los jugadores se les haya agotado su tiempo. La puntuación será igual al concurso **Normal** y ganará el que más puntos consiga.

El concurso se modela mediante la clase Contrarreloj. Está clase guarda toda la información necesaria para el seguimiento del concurso y lo ejecuta.

3.1.8. Incremento 8: Inclusión de animaciones

Llegados a este punto, los usuarios comentan que el videojuego resulta bastante aburrido, debido a que todas las imagenes son estáticas y no da sensación de interactividad. Entonces, introduzco animaciones de los personajes, del menú, y de las contestaciones correctas o incorrectas. Para ello, utilizo el programa de animación *Sinfyg*[3]. Este programa de animación me proporciona los cuadros png que necesito para realizar la emulación mediante la clase Animación, que también es diseñada e implementada.

[3]Synfig es una poderosa herramienta de animación y diseño 2D vectorial, programa opensource creado por Robert Quattlebaum con la ayuda adicional de Adrian Bentley, diseñado para producir películas animadas con pocas personas y recursos.

También, debo crearme una clase Personaje, que será la que guarde las diferentes animaciones del personaje. Esta será incluida en la clase Jugador.

Además, hay que incluir un apartado dentro de la opción de concurso número de jugadores y elección de nombre, en la que se pueda elegir el personaje que se desee.

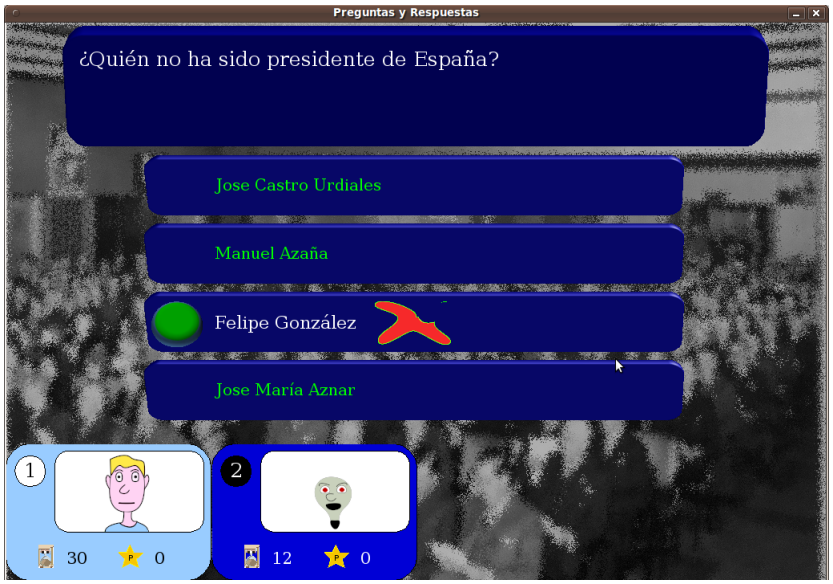

Figura 3.2: Ejemplo de animaciones

3.1.9. Incremento 9: Diseño e Implentación Concurso Tonto el Último

En este juego la dinámica de respuesta es totalmente distinta a los anteriores y la idea es que sea para 2 o más jugadores. Todos los jugadores tienen un tiempo determinado para contestar la pregunta, pueden cambiar tantas veces quieran de respuesta, pero sólo el primero que escoja la respuesta correcta conseguirá 100 puntos. El que acierte la pregunta pero no sea el primero en contestarla tendrá 0 puntos y el que conteste mal se le restará 50 puntos.

Este concurso ha sido uno de los que más bug he tenido, ya que no es fácil controlar este sistema de elección. Me ha hecho falta muchas pruebas para dejarlo funcionando correctamente.

3.1.10. Incremento 10: Diseño e Implementación de Concurso Lanza Pregunta

Este juego es recomendable para 3 o 4 jugadores, sino no tiene mucho sentido. El juego comienza dandole el turno al Jugador 1. Este tendrá que decidir a quien lanza la pregunta que se muestra. Si el jugador al que se le manda la pregunta no responde correctamente, se le restarán 50 puntos y se pasará el turno al siguiente jugador a su derecha, así hasta que se acierte o hayan contestado todos los jugadores. En cuyo caso, pasará el turno al jugador de la derecha que elegirá de nuevo a quién pasar una nueva pregunta.

En este juego he tenido un bug importante. Todos los jugadores podían lanzar la pregunta aunque no tuviesen el turno. Este bug ya ha sido solucionado y funciona correctamente.

3.1.11. Incremento 11: Gestión de Records

Una vez desarrollada toda la base del videojuego, en este incremento se le va a introducir la gestión de las puntuaciones de los jugadores. Se guardarán las 4 mejores puntuaciones por concurso, se indicará si se ha conseguido un nuevo record, se podrán consultar estás puntuaciones, etc. Para ello se desarrollan una nueva clase, la clase GestorRecords. Esta se encarga de cargar las puntuaciones de memoria secundaria a memoria principal, comprobar si hay un nuevo records, mantener la consistencia en el fichero de puntuaciones y pasar las puntuaciones de memoria principal a secundaria.

3.1.12. Incremento 12: Gestión de Jugadores

Una vez desarrollada la gestión de records se desarrolla la gestión de jugadores. Basicamente, se pueden crear tantos jugadores como el usuario desee. De cada jugador se guardará una serie de opciones por defecto y estadísticas sobre sus partidas. En concreto se guardará:

- Nombre.
- Personaje.
- Categoría de preguntas por defecto.
- Número de preguntas/bombas por defecto.
- Tiempo de contrarreloj por defecto.
- Número total de preguntas respondidas por el jugador.
- Número de preguntas acertadas por el jugador (entre la anterior y está se calculará un rating).
- Mejor puntuación de cada concurso.

Además se creará un fichero que servirá para saber los jugadores que han sido creados. Todo esto lo controlará la clase GestorJugadores. Esta clase permitirá:

- Crear un nuevo jugador.
- Mostrar las características de un jugador.
- Editar un jugador.
- Borrar un jugador.

Debido a este cambio, el modo de acceder a los concurso ahora es distinto. Al entrar en un concurso nos preguntan el número de jugador y por cada jugador nos dan la opción de cargar uno ya existente. Una vez elegido todos los jugadores, se ponen por defecto las opciones de juego del jugador 1. A continuación aparece un menú de concursos a escoger para el número de jugadores escogido.
También se ha modificado la clase Jugador, para que pueda procesar toda la información sobre este durante las partidas y luego pueda ser guardada en memoria secundaria.

3.1.13. Incremento 13: Diseño e Implementación del Concurso Pasa la Bomba

Este juego es bastante intuitivo para el jugador. En principio tenemos una bomba, que irá pasando de jugador a jugador conforme vayan acertando las preguntas. Si aciertan las respuestas se les irán sumando puntos y la bomba pasará al jugador de la derecha. Si por el contrario fallan, no se le restarán puntos pero la bomba seguirá con él. Al que le explote la bomba será el que pierda puntos. Así hasta que se terminen las bombas, que están definidas por el parámetro número de preguntas.

3.1.14. Incremento 14: Diseño e Implementación del Concurso Gran Concurso

Este juego es distinto a todos los demás, ya que no es un concurso por si mismo sino que utiliza los concursos ya creados para poder jugar consecutivamente a ellos, manteniendo la puntuación del anterior y consiguiendo una puntuación conjunta entre todos.

3.2. Diagrama de Gantt

Dada la extensión del diagrama de Gantt, me parece contraproducente introducirlo en la memoria. Este se puede consultar en la documentación que se adjunta con el software. En lugar del diagrama de Gantt, se muestra a continuación las respectivas tareas de los distintos incrementos que he ido realizando a lo largo del proyecto.

WBS	Nombre	Inicio	Fin	Trabajo	Duración	Asignado a
1	**Incremento 1**	**jul 13**	**sep 10**	**64d**	**60d**	Diego
1.1	Planificación global del proyecto	jul 13	sep 10	30d	60d	Diego
1.2	Realización de gráficos	jul 13	sep 10	30d	60d	Diego
1.3	Documentación sobre Latex	jul 13	jul 20	4d	8d	Diego
2	**Incremento 2**	**sep 11**	**oct 10**	**41d**	**30d**	Diego
2.1	Organización preguntas en memoria	sep 11	sep 16	3d	6d	Diego
2.2	Análisis de la clase Pregunta	sep 11	sep 14	2d	4d	Diego
2.3	Implementación de la clase Pregunta	sep 15	sep 16	1d	2d	Diego
2.4	Análisis de la clase MultiOpcion	sep 17	sep 20	2d	4d	Diego
2.5	Implementación de la clase MultiOpcion	sep 21	sep 22	1d	2d	Diego
2.6	Análisis de la clase MultiRespuesta	sep 17	sep 20	2d	4d	Diego
2.7	Implementación de la clase MultiRespuesta	sep 21	sep 22	1d	2d	Diego
2.8	Análisis de la clase Corta	sep 17	sep 20	2d	4d	Diego
2.9	Implementación de la clase Corta	sep 21	sep 22	1d	2d	Diego
2.10	Análisis de la clase Numerica	sep 17	sep 20	2d	4d	Diego
2.11	Implementación de la clase Numerica	sep 21	sep 24	2d	4d	Diego
2.12	Análisis de la clase TrueFalse	sep 17	sep 20	2d	4d	Diego
2.13	Implementación de la clase TrueFalse	sep 21	sep 22	1d	2d	Diego
2.14	Análisis de la clase Relacion	sep 17	sep 20	2d	4d	Diego
2.15	Implementación de la clase Relacion	sep 21	sep 22	1d	2d	Diego
2.16	Implementación clase Lote	sep 11	sep 14	2d	4d	Diego
2.17	Análisis algoritmo preguntas	sep 11	sep 25	7d	15d	Diego
2.18	Implementación del algoritmo de preguntas	sep 26	oct 10	7d	15d	Diego

Figura 3.3: Primera tabla de tareas incluida en el diagrama Gantt

WBS	Nombre	Inicio	Fin	Trabajo	Duración	Asignado a
3	**Incremento 3**	**oct 11**	**oct 16**	**15d**	**6d**	Diego
3.1	Análisis clase Video	oct 11	oct 14	2d	4d	Diego
3.2	Implementación clase Video	oct 15	oct 16	1d	2d	Diego
3.3	Análisis clase Sonido	oct 11	oct 14	2d	4d	Diego
3.4	Implementación clase Sonido	oct 15	oct 16	1d	2d	Diego
3.5	Análisis clase Fuente	oct 11	oct 14	2d	4d	Diego
3.6	Implementación clase Fuente	oct 15	oct 16	1d	2d	Diego
3.7	Análisis clase Imagen	oct 11	oct 14	2d	4d	Diego
3.8	Implementación clase Imagen	oct 15	oct 16	1d	2d	Diego
3.9	Análisis clase Inicio	oct 11	oct 14	2d	4d	Diego
3.10	Implementación clase Inicio	oct 15	oct 16	1d	2d	Diego
4	**Incremento 4**	**oct 17**	**nov 5**	**10d**	**20d**	Diego
4.1	Análisis de las opciones principales del juego	oct 17	oct 28	6d	12d	Diego
4.2	Implementación de las opciones principales del juego	oct 29	nov 5	4d	8d	Diego

Figura 3.4: Segunda tabla de tareas incluida en el diagrama Gantt

WBS	Nombre	Inicio	Fin	Trabajo	Duración	Asignado a
5	**Incremento 5**	**nov 6**	**nov 21**	**30d**	**16d**	Diego
5.1	Planificación de los concursos	nov 6	nov 15	5d	10d	Diego
5.2	Análisis clase Jugador	nov 6	nov 11	3d	6d	Diego
5.3	Implementación clase Jugador	nov 12	nov 13	1d	2d	Diego
5.4	Análisis clase GestorConcurso	nov 6	nov 13	4d	8d	Diego
5.5	Implementación clase GestorConcurso	nov 14	nov 17	2d	4d	Diego
5.6	Análisis clase Cronometro	nov 6	nov 11	3d	6d	Diego
5.7	Implementación clase Cronometro	nov 12	nov 19	4d	8d	Diego
5.8	Análisis clase Concurso	nov 6	nov 9	2d	4d	Diego
5.9	Implementación clase Concurso	nov 10	nov 13	2d	4d	Diego
5.10	Análisis de las opciones de concurso	nov 14	nov 17	2d	4d	Diego
5.11	Implementación de las opciones de concurso	nov 18	nov 21	2d	4d	Diego
6	**Incremento 6**	**nov 22**	**dic 5**	**10d**	**14d**	Diego
6.1	Análisis concurso Normal	nov 22	nov 28	5d	7d	Diego
6.2	Implementación concurso Normal	nov 29	dic 5	5d	7d	Diego
7	**Incremento 7**	**dic 6**	**dic 19**	**10d**	**14d**	Diego
7.1	Análisis concurso Contrarreloj	dic 6	dic 12	5d	7d	Diego
7.2	Implementación concurso Contrarreloj	dic 13	dic 19	5d	7d	Diego

Figura 3.5: Tercera tabla de tareas incluida en el diagrama Gantt

WBS	Nombre	Inicio	Fin	Trabajo	Duración	Asignado a
8	**Incremento 8**	**dic 20**	**feb 1**	**21d**	**32d**	Diego
8.1	Realización de Animaciones	dic 20	ene 30	15d	30d	Diego
8.2	Análisis clase Animacion	dic 20	dic 28	2d	4d	Diego
8.3	Implementacion clase Animacion	dic 29	ene 2	1d	2d	Diego
8.4	Análisis clase Personaje	dic 20	dic 26	1d	2d	Diego
8.5	Implementación clase Personaje	dic 27	dic 28	1d	2d	Diego
8.6	Modificación opciones de concursos	ene 31	feb 1	1d	2d	Diego
9	**Incremento 9**	**feb 2**	**feb 18**	**10d**	**17d**	Diego
9.1	Análisis concurso Tonto el Último	feb 2	feb 8	5d	7d	Diego
9.2	Implementación concurso Tonto el Último	feb 9	feb 18	5d	10d	Diego
10	**Incremento 10**	**feb 19**	**mar 10**	**10d**	**20d**	Diego
10.1	Análisis concurso Lanza Pregunta	feb 19	feb 28	5d	10d	Diego
10.2	Implementación concurso Lanza Pregunta	mar 1	mar 10	5d	10d	Diego
11	**Incremento 11**	**mar 11**	**abr 2**	**11d**	**23d**	Diego
11.1	Análisis de la clase GestorRecords	mar 11	mar 25	7d	15d	Diego
11.2	Implementación de la clase GestorRecords	mar 26	mar 29	2d	4d	Diego
11.3	Modificaciones en los concursos para contabilizar puntuaciones	mar 30	abr 2	2d	4d	Diego
12	**Incremento 12**	**abr 3**	**abr 12**	**6d**	**10d**	Diego
12.1	Análisis de la clase GestorJugadores	abr 3	abr 6	2d	4d	Diego
12.2	Implementación de la clase GestorJugadores	abr 7	abr 10	2d	4d	Diego
12.3	Modificación de la clase Jugador	abr 11	abr 11	1d	1d	Diego
12.4	Modificación de la estructura de elección de concursos	abr 12	abr 12	1d	1d	Diego

Figura 3.6: Cuarta tabla de tareas incluida en el diagrama Gantt

WBS	Nombre	Inicio	Fin	Trabajo	Duración	Asignado a
13	**Incremento 13**	**abr 13**	**abr 27**	**7d**	**15d**	Diego
13.1	Análisis concurso Pasa la Bomba	abr 13	abr 22	5d	10d	
13.2	Implementación concurso Pasa la Bomba	abr 23	abr 27	2d	5d	
14	**Incremento 14**	**abr 28**	**may 22**	**14d**	**25d**	Diego
14.1	Análisis concurso Gran Concurso	abr 28	may 7	7d	10d	
14.2	Implementación concurso Gran Concurso	may 8	may 22	7d	15d	
15	Realización de la memoria de proyecto	jul 13	jun 1	30d	312d	Diego
16	Busqueda y actualización de preguntas	jul 13	jun 1	30d	312d	Diego

Figura 3.7: Quinta tabla de tareas incluida en el diagrama Gantt

3.3. Esfuerzo

Hay que destacar que la mayor parte del tiempo se ha dedicado a pensar, analizar los distintos concursos que se podían hacer. Otra parte importante ha sido la de optimización de código y la administración correcta de los recursos que necesita el software.

La parte más complicada de implementar ha sido la de transformación de las preguntas en fichero .gift a preguntas en memoria. Esto ha requerido de un algoritmo secuencial en el que hay que considerar todas las opciones del fichero posibles, así como filtrar la información que contenía el fichero.

También me ha supuesto mucho trabajo las pruebas de los diferentes concurso, ya que se hace pesado tener que estar ejecutando una y otra vez el videojuego para comprobar errores y probar todos los casos posibles.

La gestión por fichero de la puntuación y los jugadores también ha requerido un gran conjunto de pruebas para verificar que funciona en todos los casos posibles.

La parte de diseño tambien ha sido tediosa. Tener que haber hecho la mayoría de las imagenes que aparecen en el concurso es un desafio para una persona como yo, que no tiene un gran sentido artístico. Esto me ha hecho valorar mucho el trabajo de los diseñores gráficos.

Capítulo 4

Análisis

4.1. Metodología de Desarrollo

En este proyecto se ha seguido una metodología de desarrollo ágil de software, que se basa en procesos ágiles, es decir, se intenta evitar los formales caminos de las metodologías tradicionales enfocándose en las personas y los resultados.

Esta metodología promueve iteraciones en el desarrollo a lo largo de la vida del proyecto, minimizando los riesgos al desarrollar software en cortos períodos de tiempo. Estos períodos de tiempo se llaman iteraciones, las cuales pueden durar de una a cuatro semanas. Cada iteración debe de contener: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener una versión de prueba al final de cada iteración. Cada vez que se finaliza una iteración, el proyecto debe pasar a manos de los revisores, que en mi caso puede ser cualquier persona ajena al desarrollo del proyecto y lo evalúa, comentan las cosas que más le ha llamado la atención y los fallos más evidentes, y aconseja cambios en cualquiera de las funcionalidades que se han implantado en esta nueva iteración.

Estos métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. Lo cual es más rápido, consigue un producto más cercano al público y en menor tiempo.

Como hemos ido viendo, el proyecto comenzó con sólo el aspecto visual y a medida que fueron avanzando las iteraciones o incrementos se fueron añadiendo nuevas funcionalidades y concursos, hasta conseguir un producto final con varios concursos y muchas funcionalidades. Siempre teniendo en cuenta las opiniones de las distintas personas, que iban adquiriendo el papel de revisores, al no disponer de un equipo de trabajo, como es mi caso, han sido personas cercanas, como mi familia y amigos.Estos revisaban las versiones después de cada iteración y comentaban sus impresiones.

4.2. Análisis del sistema

El objetivo del análisis de sistemas es realizar un **Documento de Especificación de Requisitos**: especificar qué tiene que hacer el sistema y no cómo desarrollarlo.

Para el análisis de nuestro sistema vamos a usar un método de análisis orientado a objetos.

Para desarrollar las ideas para nuestro proyecto hemos usado la técnica llamada **Brainstorming** (tormenta de ideas) en la cuál yo me reunía con algún miembro de mi familia o algún amigo y comentaba el tipo de concurso que quería realizar, cada uno aportaba sus ideas sin juzgar su validez yo las valoraba, y me quedaba con las ideas más interesantes. Una vez que tenía las ideas más interesantes, dedicaba un tiempo a valorar las ideas más originales y que se adaptaban mejor a la naturaleza del proyecto y a sus características.

4.3. Funciones del Producto

Prerres está formado por varios tipos de clases. Primero veremos las funciones principales de las clases de la interfaz gráfica y luego describiremos cada uno de los concursos que componen el producto final.

4.3.1. Interfaz gráfica

Uno de los pilares de un videojuego, sea del tipo que sea, es la interfaz gráfica y el juego que nos ocupa no es ninguna excepción.
El videojuego dispone de una interfaz agradable y sencilla, utilizando los mandos buzzers. Esto hace que la forma de seleccionar una opción, una respuesta, etc, se haga con una sola pulsación de botón. Además los contenidos están bien organizados y con posible retorno en los menús con la opción volver. Aunque la interfaz es sencilla, creo que esto ayuda a que todo tipo de jugadores puedan acceder al juego de forma fácil, sin tener demasiada soltura en el manejo de un dispositivo informático.

Además, la idea de utilizar siempre mandos es muy importante para utilizar este videojuego , ya que si hubiese que utilizar otros dispositivos de entrada como ratón o teclado, el juego se relentizaría mucho y no podría realizarse entre todos, sino alguien tendría que tener el control de la máquina mientras los demás sólo miran. Aún así, es posible manejar el juego por medio de teclado aunque no es lo recomendable.

Posiblemente se podría mejorar bastante los gráficos del juego para hacerlo más atractivo, pero pienso que es un proyecto bastante largo y que hay que abarcar todos los campos con un tiempo reducido. Cabe destacar que cualquier juego o producto software de la actualidad está desarrollado usando un equipo de trabajo bien organizado y que cada miembro se ocupa de una parte del software, y que en mi caso, soy yo el que está a cargo tanto del desarrollo de ideas para el juego como de la implementación, pasando por otros campos como el diseño gráfico.

4.3.2. Biblioteca de preguntas

Una de las funciones más importantes del videojuego es la de biblioteca de preguntas en formato GIFT.

Esta opción nos permite almacenar todo tipo de preguntas en formato GIFT en memoria principal, para utilizarlo en nuestro videojuego. En mi caso, sólo se utilizan las preguntas de tipo MultiOpción, pero esta biblioteca está diseñada para poder utilizar todos los tipos de preguntas de igual manera que se utilizan las de MultiOpción en mi proyecto. Para más información sobre la biblioteca de pregunta consultar el apéndice D.

Esta tarea además, la hace de forma transparente al usuario mediante un script en perl y un algoritmo en C++. Esto ayuda al usuario a la hora de introducir preguntas al videojuego, ya que sería tedioso tener que ir introduciendo una a una las preguntas con las que queremos incrementar la biblioteca de preguntas del videojuego.

Incluso se pueden actualizar las preguntas que hay en memoria principal mientras la aplicación está en uso. Unicamente habría que modificar el archivo correspondiente con las preguntas deseadas y utilizar la opción Actualizar Preguntas. Todo esto se explicará más detalladamente en el apéndice C.

4.3.3. Gestión de Jugadores

Para darle una sensación de continuidad y de reto, se crea la gestión de jugadores. Esto nos permite crearnos un jugador y jugar con el siempre que utilicemos el juego. Con esto conseguiremos estadísiticas de nuestras partidas y nuevos retos a conseguir.

La gestión de jugadores incluye: crear, editar, mostrar y borrar jugadores.

4.3.4. Gestión de Records

Aparte de la gestión de jugadores, el juego también dispone de una gestión de las puntuaciones conseguidas en cada concurso. Simplemente guardará las mejores puntuaciones de cada concurso y se podrán consultar. Esto supone un reto para los jugadores del juego, ya que intentarán superar las mejores puntuaciones conseguidas por otros usuarios.

4.3.5. Concurso Normal

Una vez que hemos hablado de las funciones que realiza el núcleo del juego, hablaremos de los distintos concursos.
El **Concurso Normal** se trata de un concurso sencillo en el que básicamente hay que contestar preguntas y acertar más que los demás jugadores.

El juego comienza con el turno para el jugador 1, que tendrá 30 segundos para contestar la pregunta que se le formula. Si contesta correctamente se le sumarán 100 puntos o menos, dependiendo del tiempo de respuesta, a su marcador. En caso de contestar incorrectamente o en caso de que el tiempo se acabe, se le restará 50 puntos a su marcador. Las puntuaciones nunca pueden ser negativas, así que en caso de fallar y tener 0 puntos, el marcador seguirá tal cual. En cualquiera de los dos casos, el turno pasa al jugador que está a su derecha, que dispone del mismo tiempo para contestar otra pregunta. Este tiempo

se reinicia para cada pregunta o lo que es lo mismo, cada jugador tiene 30 segundos para contestar cada pregunta que se les formule.

Una vez que termina el turno del último jugador, el turno vuelve a pasar al primer jugador y así hasta que todos hayan respondido el número de preguntas seleccionado en las opciones.

4.3.6. Concurso Contrarreloj

Este concurso como da a entender su título consiste en una contrarreloj. Los jugadores tendrán un tiempo determinado en las opciones de concurso que irá transcurriendo conforme responden preguntas. Una vez acabado el tiempo de todos los jugadores se contabiliza el resultado.

Una vez seleccionado todo empieza el primer jugador. Se le muestra una pregunta y la debe contestar en el menor tiempo posible. Una vez conteste la pregunta se evaluará si es correcta o incorrecta actualizando en correspondencia su marcador[1] (correcta 100 puntos o menos, depediendo del tiempo de respuesta, incorrecta -50 puntos) y se guardará ese tiempo como el tiempo que le queda para responder a las demás preguntas. Esto ocurrirá para cada uno de los jugadores del concurso y habrá que esperar que todos acaben su tiempo para saber quien es el ganador. Si el tiempo termina y no se ha contestado a la pregunta, no tendrá penalización en el marcador.

4.3.7. Concurso Pasa la Bomba

Este concurso es bastante intuitivo. Una bomba va pasando de un jugador a otro hasta que explota.

El concurso comienza con el turno para el jugador 1, y empieza a contar el tiempo para que explote la bomba. El jugador deberá contestar la pregunta correctamente para poder pasar la bomba, sino seguirá teniendo la bomba en su poder. Al acertar una pregunta se le sumaran 10 puntos o menos dependiendo del tiempo de respuesta y al fallar no se quitarán puntos, simplemente se quedarán con la bomba. Al que le explote la bomba perderá 50 puntos. Habrá tantas bombas como número de preguntas tenga el concurso en las opciones.

4.3.8. Concurso Tonto el Último

Este concurso tiene una dinámica diferente al anteriormente presentado.

Cada jugador dispone de 15 segundos para contestar de forma simultanea a la pregunta que se le formula. Pueden cambiar tantas veces quieran de respuesta pero sólo el primero que seleccione la respuesta correcta será el que se lleve los 100 puntos. En el caso de contestar correctamente pero no ser el primero, no restará ni sumará puntos al marcador. En cualquier otro caso, se le restarán 50 puntos al marcador.

Una vez contestada y verificada la respuesta de una pregunta se mostrará otra y se actuará de la misma forma. Así sucesivamente hasta que se contesten al número de preguntas seleccionado en las opciones de concurso.

[1] Indistintamente del tipo de concurso que se juegue, siempre se mostrará la respuesta correcta a la pregunta que se formula.

4.3.9. Concurso Lanza Pregunta

Este es el último concurso que se incluye en el apartado de **Concursos Simples** y se trata del **Concurso Lanza Pregunta**. Este concurso como bien se puede entender por su título, consiste en lanza una pregunta a cualquiera de los equipos participantes. Este concurso está indicado para 3 o 4 jugadores. Las opciones de concursos son identicas a las anteriores por lo tanto, pasamos a describir el funcionamiento del mismo.

El concurso empieza con el turno para el jugador 1 que debe elegir a que jugador le lanza la pregunta que se le muestra en pantalla. Una vez seleccionado el jugador al que se le lanza la pregunta, este tiene 20 segundos para contestar la pregunta de forma correcta. En caso de que se conteste correctamente se le adjudicará 100 puntos o menos, dependiendo del tiempo de respuesta, al marcador. En caso contrario, se le restarán 50 puntos al marcador y el turno pasará al jugador que tiene a su derecha en forma de rebote. Así sucesivamente hasta que se acierte la pregunta o todos los jugadores hayan contestado y ninguno acierte la pregunta. Nuevamente se formulará una nueva pregunta y el jugador que tenga el turno deberá decidir a quien pasar la pregunta. Así sucesivamente hasta que se formulen el número de preguntas que se seleccionó en las opciones de concurso.

4.4. Diagrama de Casos de Uso

A continuación mostraremos los casos de uso con sus descripciones.

4.4.1. Caso de Uso: Jugar a un Concurso

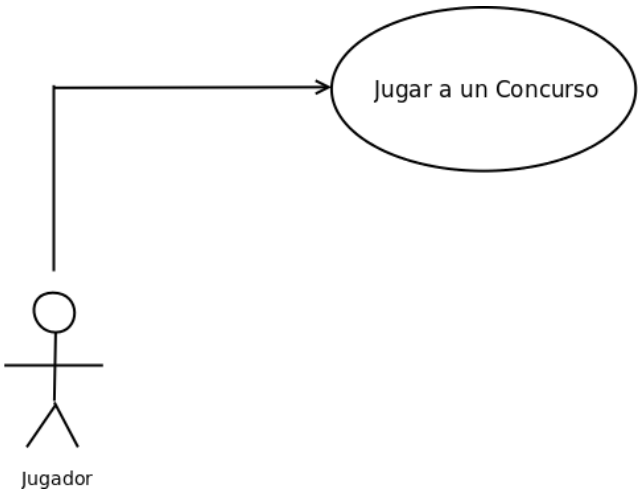

Figura 4.1: Diagrama de Caso de Uso Jugar a un Concurso

Descripción del Caso de Uso: *Jugar a un Concurso*

Descripción: El jugador juega a uno de los concursos del videojuego.
Actor: Jugador/es.
Resumen: El jugador elige y juega a un concurso y el sistema registra la partida y muestra las puntuaciones y ganadores.
Escenario principal

1. El *Jugador* quiere jugar a un concurso.
2. El *Sistema* muestra el menú de número de jugadores del concurso.
3. El *Jugador* elige el número de jugadores.
4. El *Sistema* pide al usuario que carge los jugadores mediante un menú.
5. El *Sistema* muestra un menú con los concursos que pueden jugar ese número de jugadores.
6. El *Jugador* elige el concurso al que desea jugar.
7. El *Sistema* muestra el menú del concurso.
8. El *Jugador* elige jugar al concurso.
9. El *Sistema* incia el concurso.
10. El/Los *Jugador/es* y *Sistema* interactuan para jugar al concurso.
11. El *Sistema* muestra la puntuación parcial del concurso.
12. El *Jugador* decide continuar.
13. El *Sistema* muestra la puntuación final del concurso.
14. El *Sistema* comprueba los records y los registra y muestra por pantalla.
15. El *Sistema* actualiza la información de los jugadores despues de jugar.
16. El *Jugador* decide volver al menú principal.

Escenario alternativo
*a. En cualquier momento un jugador aborta el concurso.
    *a1. El *Sistema* cierra el concurso y carga de nuevo el menú de concursos.

4a. No hay tantos jugadores disponibles.
    4a1. El *Sistema* muestra un mensaje al usuario y vuelve al paso 2.

7a. El *Jugador* elige cambiar las opciones del concurso.
    7a1. El *Sistema* muestra el menú de opciones del concurso.
    7a2. El *Jugador* elige una de las opciones.
    7a3. El *Sistema* muestra los valores disponibles para esa opción.
    7a2. El *Jugador* elige uno de los valores.

4.4.2. Caso de Uso: Gestión de Jugadores

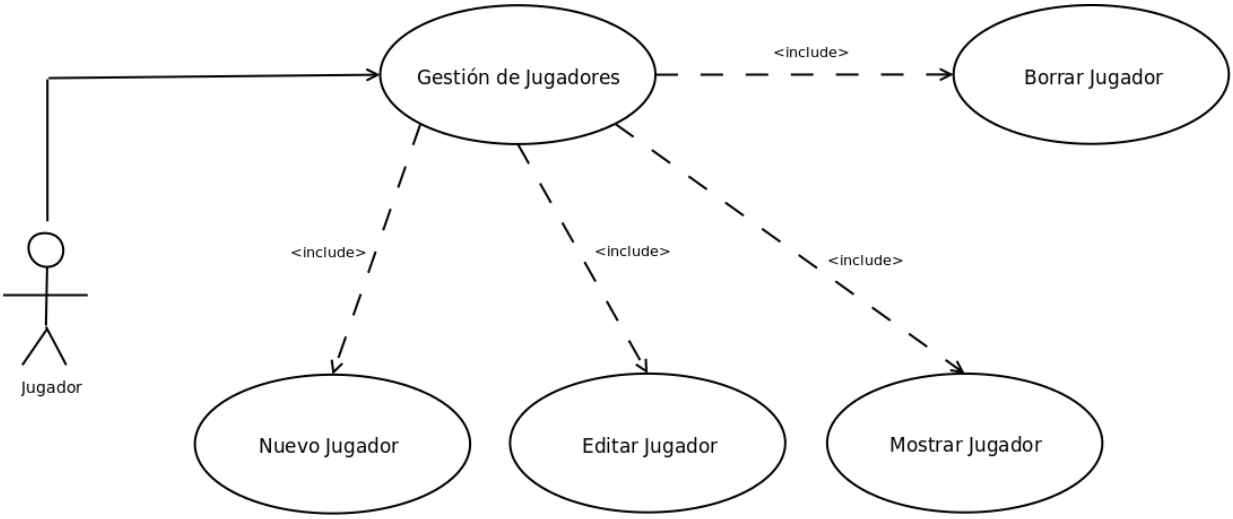

Figura 4.2: Diagrama de Caso de Uso Gestión de Jugadores

Descripción del Caso de Uso: *Gestión de Jugadores*
Descripción: Gestionar los jugadores del videojuego.
Actor: Jugador.
Resumen: Gestinar los jugadores del videojuego. Pudiendo crear, editar, mostrar o borrar un jugador.
Escenario principal

1. El *Jugador* accede a la sección "Gestión de Jugadores" del menú principal.
2. El *Sistema* muestra el menú de gestión de jugadores.
3. El *Jugador* elige una opción.
4. El *Jugador* vuelve al menú principal del videojuego.

Escenario alternativo
*a. En cualquier momento un jugador aborta el concurso.
*a1. El *Sistema* cierra el concurso y carga de nuevo el menú de principal.

3a. El jugador elige crear un jugador nuevo.
3a1. Include Nuevo Jugador.

3b. El jugador elige editar un jugador.
3b1. Include Editar Jugador.

3c. El jugador elige mostrar un jugador.
3c1. Include Mostrar Jugador.

3d. El jugador elige mostrar un jugador.
3d1. Include Borrar Jugador.

4.4.3. Caso de Uso: Nuevo Jugador

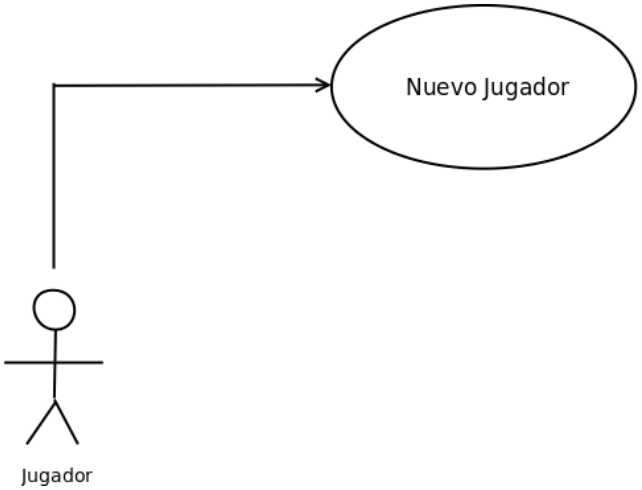

Figura 4.3: Diagrama de Caso de Uso Nuevo Jugador

Descripción del Caso de Uso: *Nuevo Jugador*
Descripción: El jugador crea un nuevo jugador.
Actor: Jugador.
Resumen: El jugador quiere crea un nuevo jugador y el sistema le pide nombre, personaje, categoría de las preguntas por defecto, número de preguntas por concurso por defecto y tiempo de contrarreloj por defecto.

Escenario principal

1. El *Jugador* quiere crea un nuevo jugador.
2. El *Sistema* muestra el menú para escribir el nombre del nuevo jugador.
3. El *Jugador* introduce el nombre del nuevo jugador.
4. El *Sistema* comprueba que no exista el jugador.
5. El *Sistema* muestra el menú de personajes del videojuego.
6. El *Jugador* elige uno de los personajes para representarlo.
7. El *Sistema* muestra el menú de categorías del juego.
8. El *Jugador* elige su categoría de preguntas por defecto.
9. El *Sistema* muestra el menú de número de preguntas del concurso por defecto.
10. El *Jugador* elige el número de preguntas del concurso por defecto entre las opciones.
11. El *Sistema* muestra el menú de tiempo de contrarreloj por defecto.
12. El *Jugador* elige el tiempo de contrarreloj por defecto entre las opciones.
13. El *Sistema* crea un nuevo jugador.

14. El *Sistema* muestra un mensaje al jugador indicandole que todo ha ido correctamente.

Escenario alternativo
*a. En cualquier momento un jugador puede volver al paso anterior.

4a. El jugador ya existe.
    4a1. El *Sistema* lo informa por pantalla.
    4a2. Vuelve al paso 2.

4.4.4. Caso de Uso: Editar Jugador

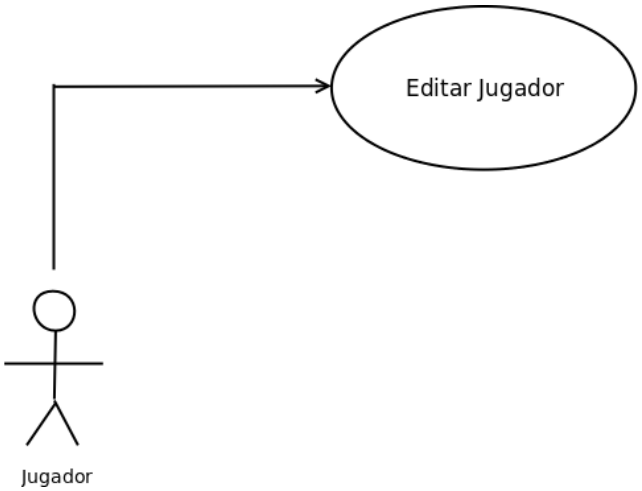

Figura 4.4: Diagrama de Caso de Uso Editar Jugador

Descripción del Caso de Uso: *Editar Jugador*
Descripción: El jugador edita un jugador ya existente.
Actor: Jugador.
Resumen: El jugador quiere editar alguna de las características de un jugador ya existente y el sistema le muestra las opciones disponibles. El jugador puede cambiar el nombre, personaje, categoría de preguntas por defecto, número de preguntas del concurso por defecto o tiempo de contrarreloj. por defecto.

Escenario principal

1. El *Jugador* quiere editar un jugador ya existente.
2. El *Sistema* muestra el menú de características a editar del jugador.
3. El *Jugador* el jugador elige una opción.
4. El *Jugador* decide volver al menú de gestión de jugadores.

Escenario alternativo
*a. En cualquier momento un jugador puede volver al paso anterior.

3a. El jugador elige cambiar el nombre del jugador.
    3a1. El *Sistema* muestra el menú para escribir el nombre del nuevo jugador.
    3a2. El *Jugador* introduce el nombre del nuevo jugador.
    3a3. El *Sistema* comprueba que no exista el jugador.
    3a4. Si existe el *Sistema* lo informa por pantalla y vuelve al paso a1.

3b. El jugador elige cambiar el personaje que representa al jugador.
    3b1. El *Sistema* muestra el menú de personajes del videojuego.
    3b2. El *Jugador* elige un nuevo personaje que representará al jugador.

3c. El jugador elige cambiar la categoría de las preguntas por defecto del jugador.
    3c1. El *Sistema* muestra el menú de categorías del juego.
    3c2. El *Jugador* elige su categoría de preguntas por defecto.

3d. El jugador elige cambiar el número de preguntas del concurso por defecto.
    3d1. El *Sistema* muestra el menú de número de preguntas del concurso.
    3d2. El *Jugador* elige el número de preguntas del concurso por defecto entre las opciones.

3e. El jugador elige cambiar el tiempo de contrarreloj por defecto del jugador.
    3e1. El *Sistema* muestra el menú de tiempo de contrarreloj del concurso.
    3e2. El *Jugador* elige el tiempo de contrarreloj por defecto entre las opciones.

4.4.5. Caso de Uso: Mostrar Jugador

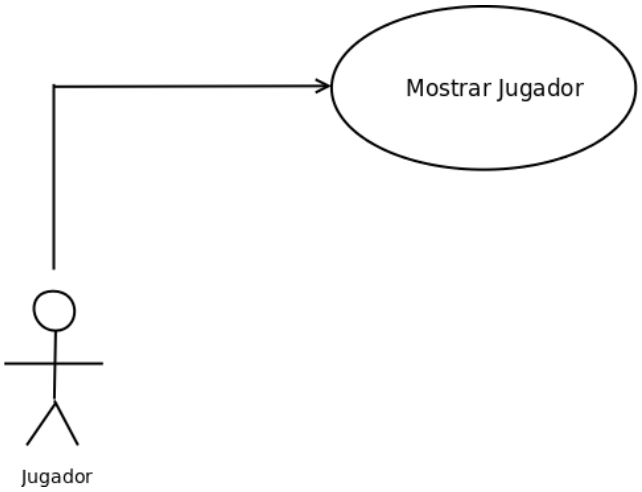

Figura 4.5: Diagrama de Caso de Uso Mostrar Jugador

Descripción del Caso de Uso: *Mostrar Jugador*
Descripción: El jugador quiere ver las características de un jugadores ya existente.
Actor: Jugador.
Resumen: El jugador quiere ver las características de un jugador ya existente. El sistema las muestra por pantalla en forma de menús.
Escenario principal

1. El *Jugador* quiere ver las características de un jugador ya existente.

2. El *Sistema* muestra el menú de características del jugador.

3. El *Jugador* el jugador elige una opción.

4. El *Jugador* decide volver al menú de gestión de jugadores.

Escenario alternativo
3a. El jugador elige la opción información sobre el jugador.
    3a1. El *Sistema* muestra el número de preguntas respondidas por el jugador y el rating de respuestas correctas.

3b. El jugador elige la opción opciones por defecto del jugador.
    3b1. El *Sistema* muestra las opciones de concurso por defecto del jugador.

3c. El jugador elige la opción mejores puntuaciones del jugador.
    3c1. El *Sistema* muestra la mejor punutación en cada concurso del jugador.

4.4.6. Caso de Uso: Borrar Jugador

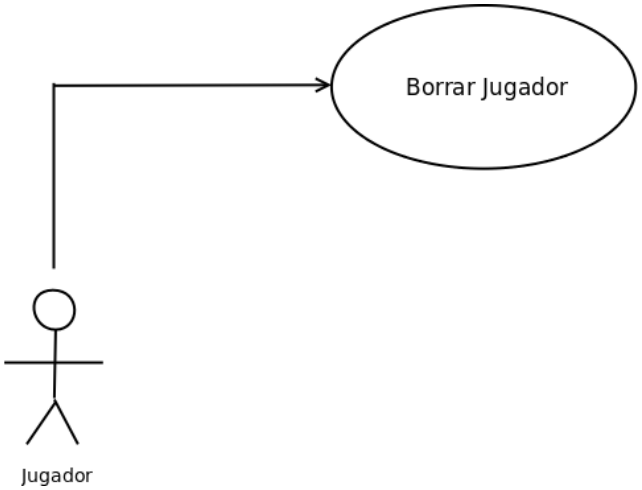

Figura 4.6: Diagrama de Caso de Uso Borrar Jugador

Descripción del Caso de Uso: *Borrar Jugador*
Descripción: El jugador quiere borrar un jugador existente.
Actor: Jugador.
Resumen: El jugador quiere borrar un jugador existente, el sistema le pide confirmación y si el jugador está de acuerdo el sistema borra al usuario.

Escenario principal

1. El *Jugador* quiere borrar un jugador existente.

2. El *Sistema* muestra los jugadores existentes en el videojuego.

3. El *Jugador* elige uno de los jugadores.

4. El *Sistema* pide confirmación al jugador para borrar al jugador.

5. El *Sistema* borra jugador al jugador y tambien elimina sus puntuaciones en los records.

Escenario alternativo
4a. El jugador se arrepiente y da marcha atras.
    4a1. Vuelve al paso 2.

4.5. Diagrama de Clases

Debido a la extensión del diagrama se ha tenido que separar algunas partes del mismo. A continuación se muestra el diagrama de clases y seguidamente las partes de la biblioteca gift y los concursos.

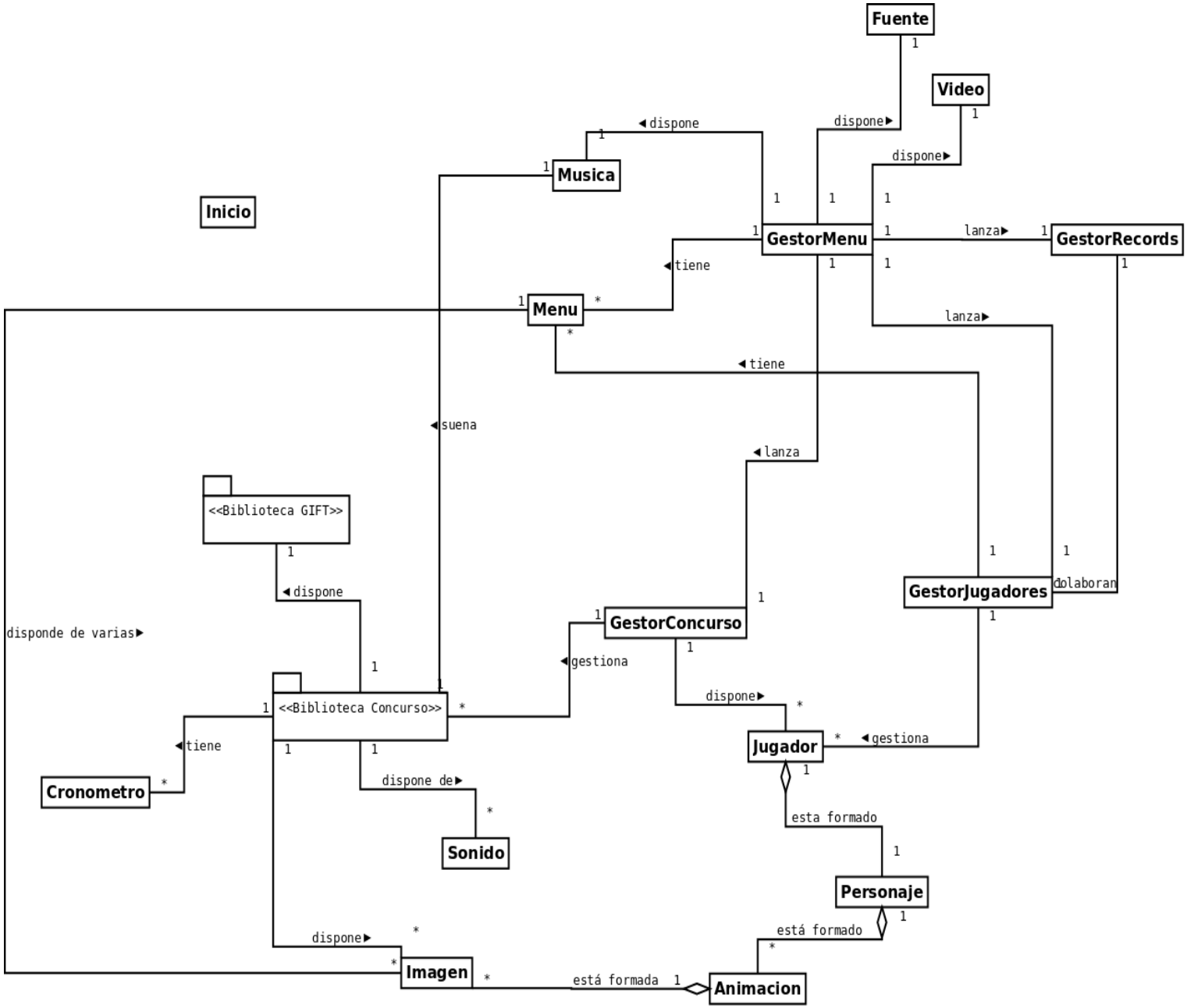

Figura 4.7: Diagrama de Clases

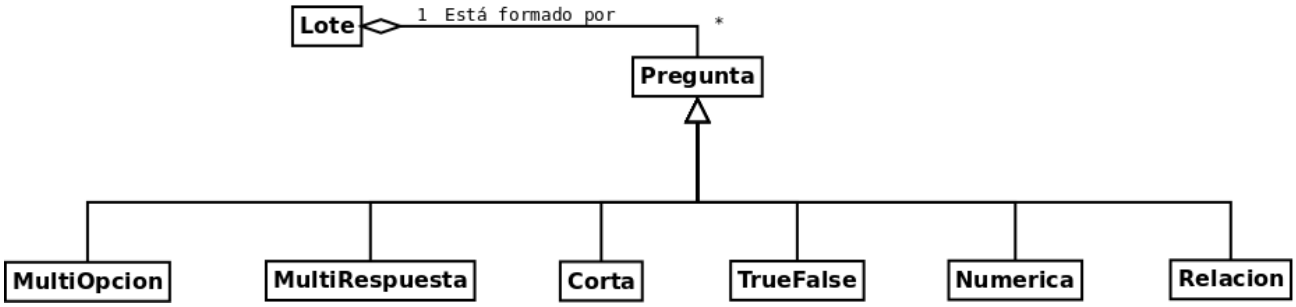

Figura 4.8: Diagrama de Clases Biblioteca GIFT

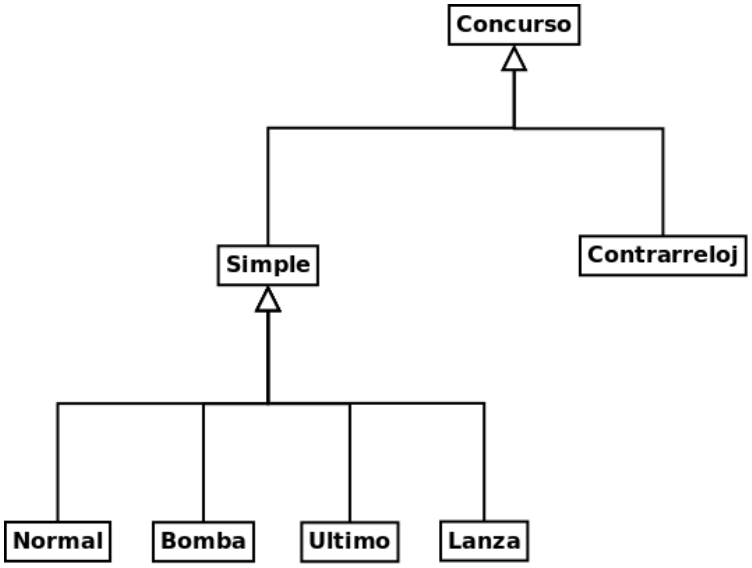

Figura 4.9: Diagrama de Clases Biblioteca Concurso

En el diagrama de clases general vemos que la clase central es **GestorMenu** ya que es la que controla todos los lanzamientos de juegos y carga los diferentes menús de la aplicación. Otro papel muy importante lo recibe la clase **Menu** ya que es la encargada de mostrar y seleccionar los distintos menús que componen el juego.También la clase **GestorConcurso** juega un papel importante, ya que es la encargada de configurar y ejecutar cada uno de los diferentes concursos. De ahí que tenga tantas relaciones con las distintas clases del videojuego.

Hemos decidido crear una clase por cada concurso, que hereda de la clase Concurso que es la que tiene todas las variables comunes a todos los concursos.

Para más información consultar tanto en la documentación como en el capítulo 5.

Dado que hay que guardar todas las opciones del formato gift, las clases de las preguntas tienen muchos atributos. Esto podría parecer un inconveniente, pero gracias a esto cualquiera puede reutilizar este código para utilizar las preguntas de gift con sus pesos, comentarios, nombres, etc.

Capítulo 5

Diseño

El proceso de Diseño se puede definir como una actividad consistente en aplicar distintas técnicas y principios con la finalidad de definir un sistema con suficiente detalle para que se pueda implementar.

Después de un gran trabajo de análisis y tener todos el análisis de requisitos de todos los concursos, la tarea de diseño es más liviana. Sólo tenemos que pensar en cómo lo hace la máquina, definir las operaciones más importante y una vez hecho esto pasar a la fase de implementación.

De manera breve comentaremos las distintas funciones y procedimientos que componen los distintos módulos, las distintas opciones que nos permiten y lo que nos aporta.

Antes, se muestra los distintos diagramas de pantallas que habrá el videojuego:

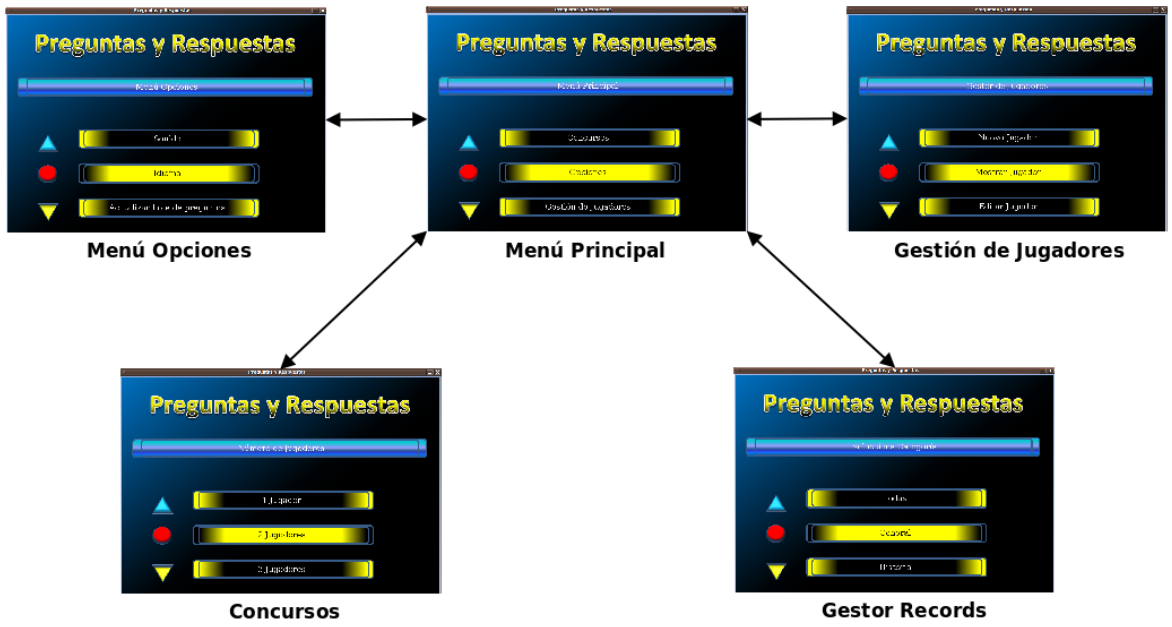

Figura 5.1: Diagrama de pantallas Menú Principal

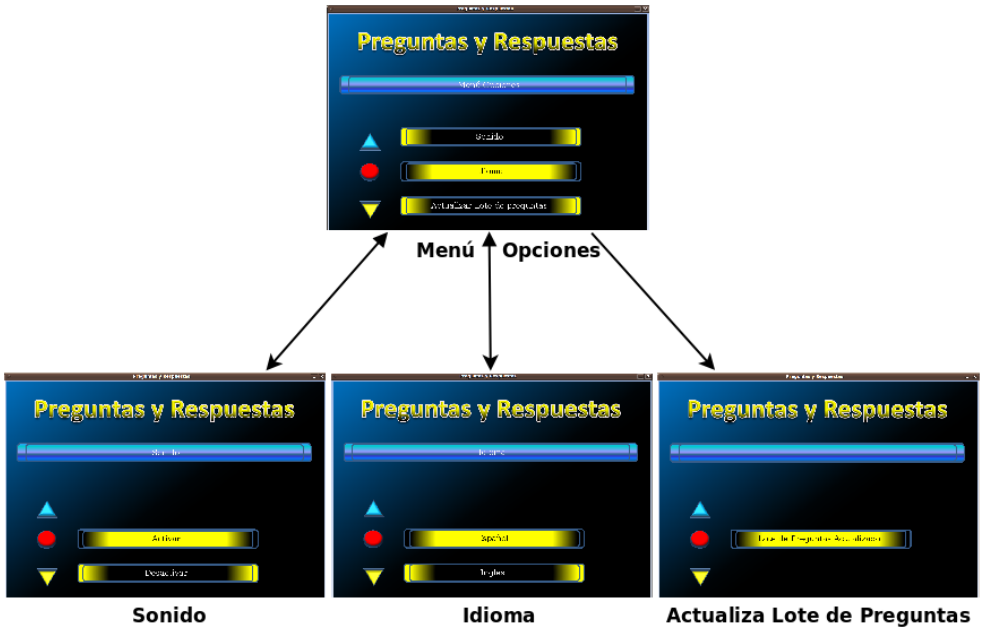

Figura 5.2: Diagrama de pantallas Menú Opciones

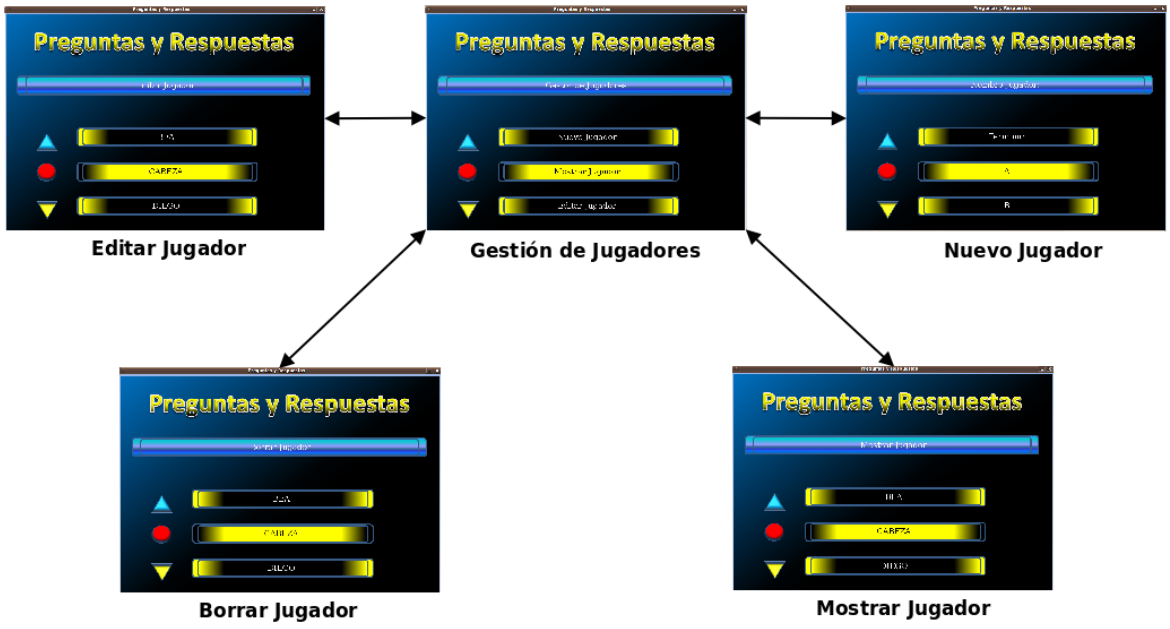

Figura 5.3: Diagrama de pantallas Gestor Jugadores

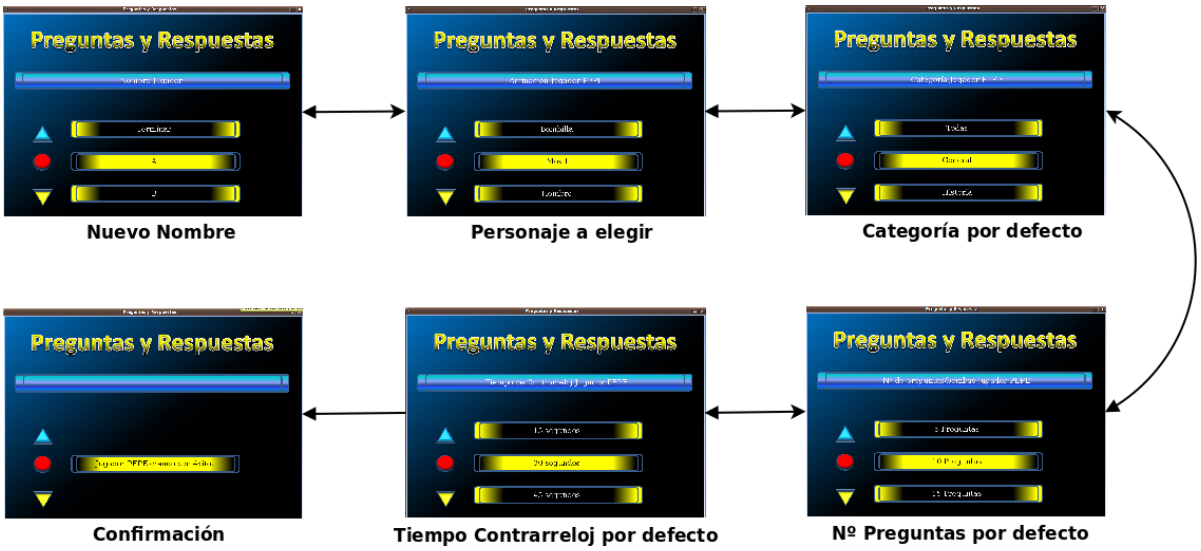

Figura 5.4: Diagrama de pantallas Nuevo Jugador

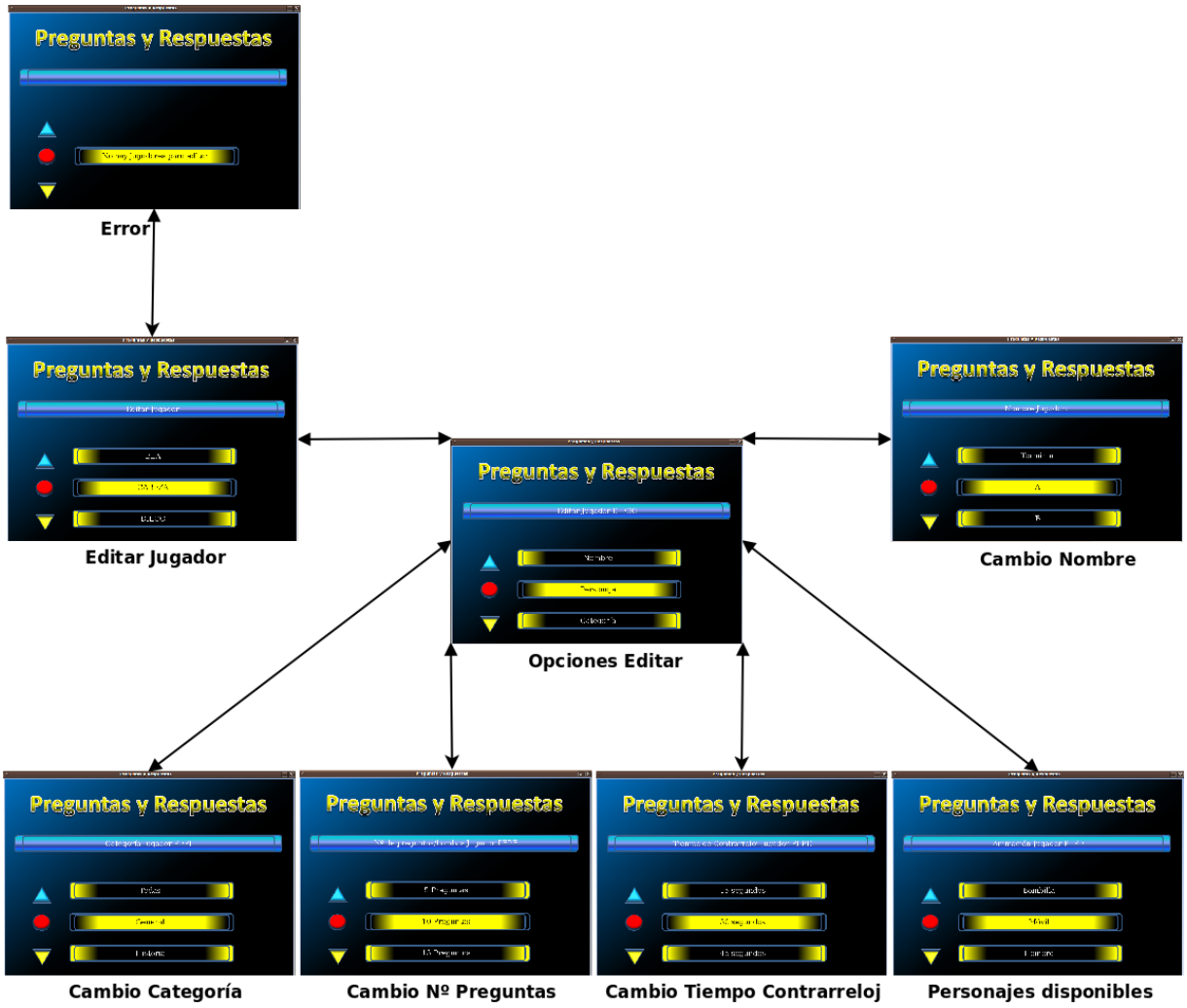

Figura 5.5: Diagrama de pantallas Editar Jugador

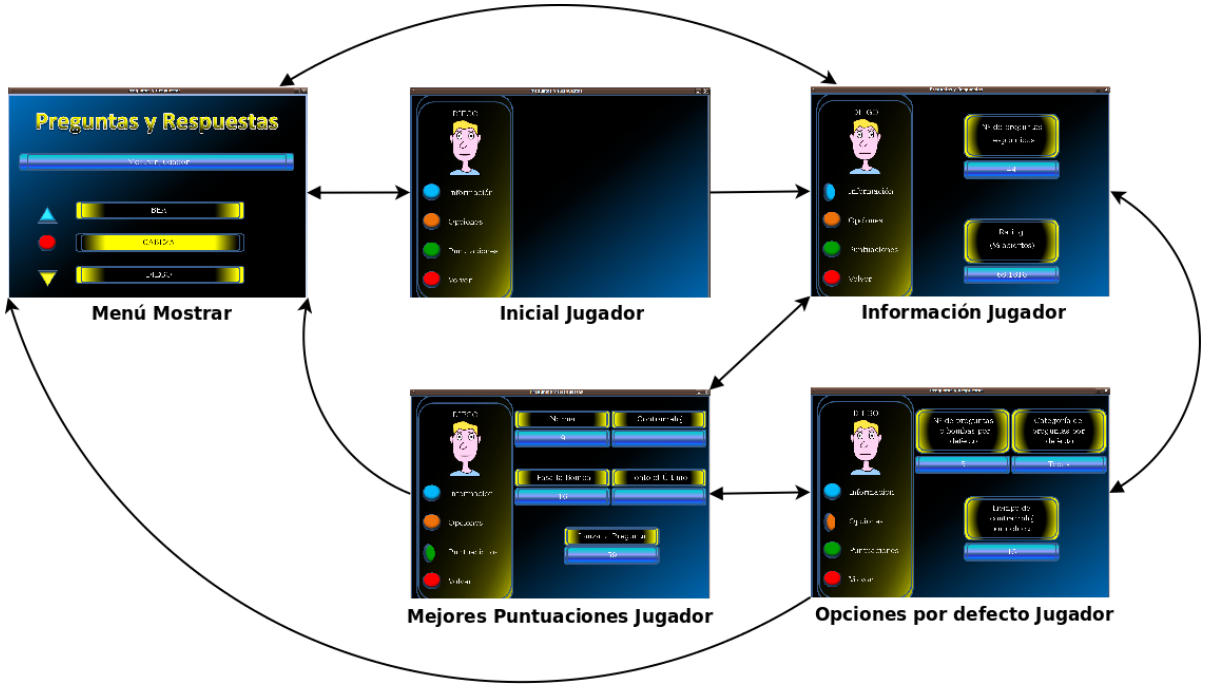

Figura 5.6: Diagrama de pantallas Mostrar Jugador

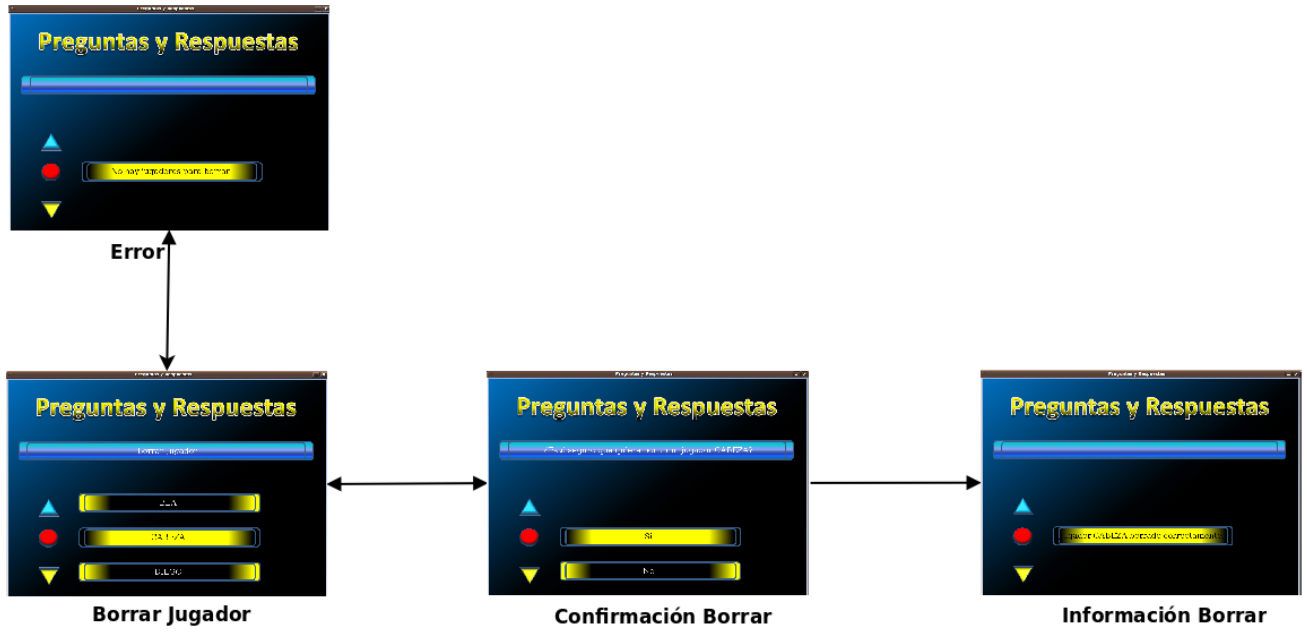

Figura 5.7: Diagrama de pantallas Borrar Jugador

Figura 5.8: Diagrama de pantallas Gestor Records

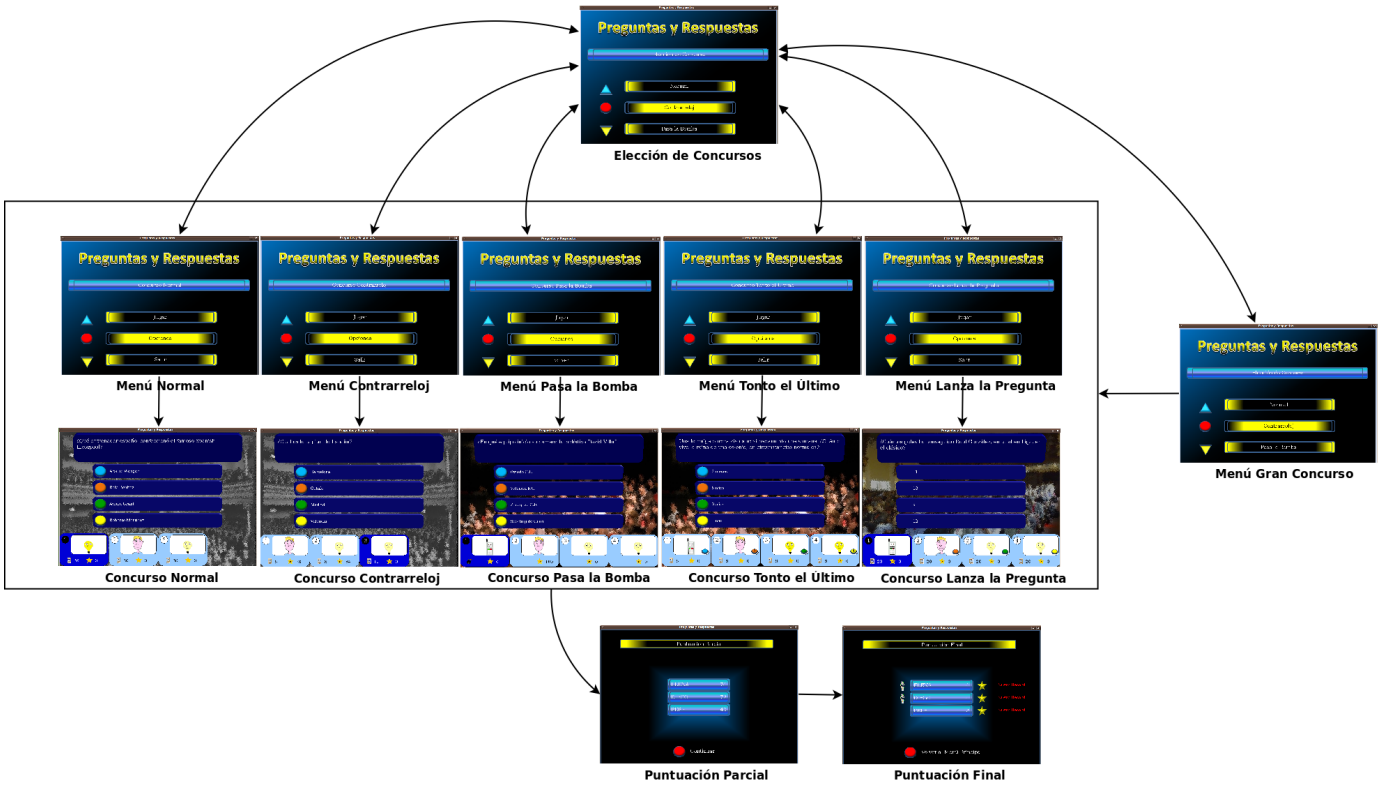

Figura 5.9: Diagrama de pantallas Concursos

Una vez expuestos los diagramas de pantallas se pasa a detallar el diagrama de clases, con sus operaciones y atributos:

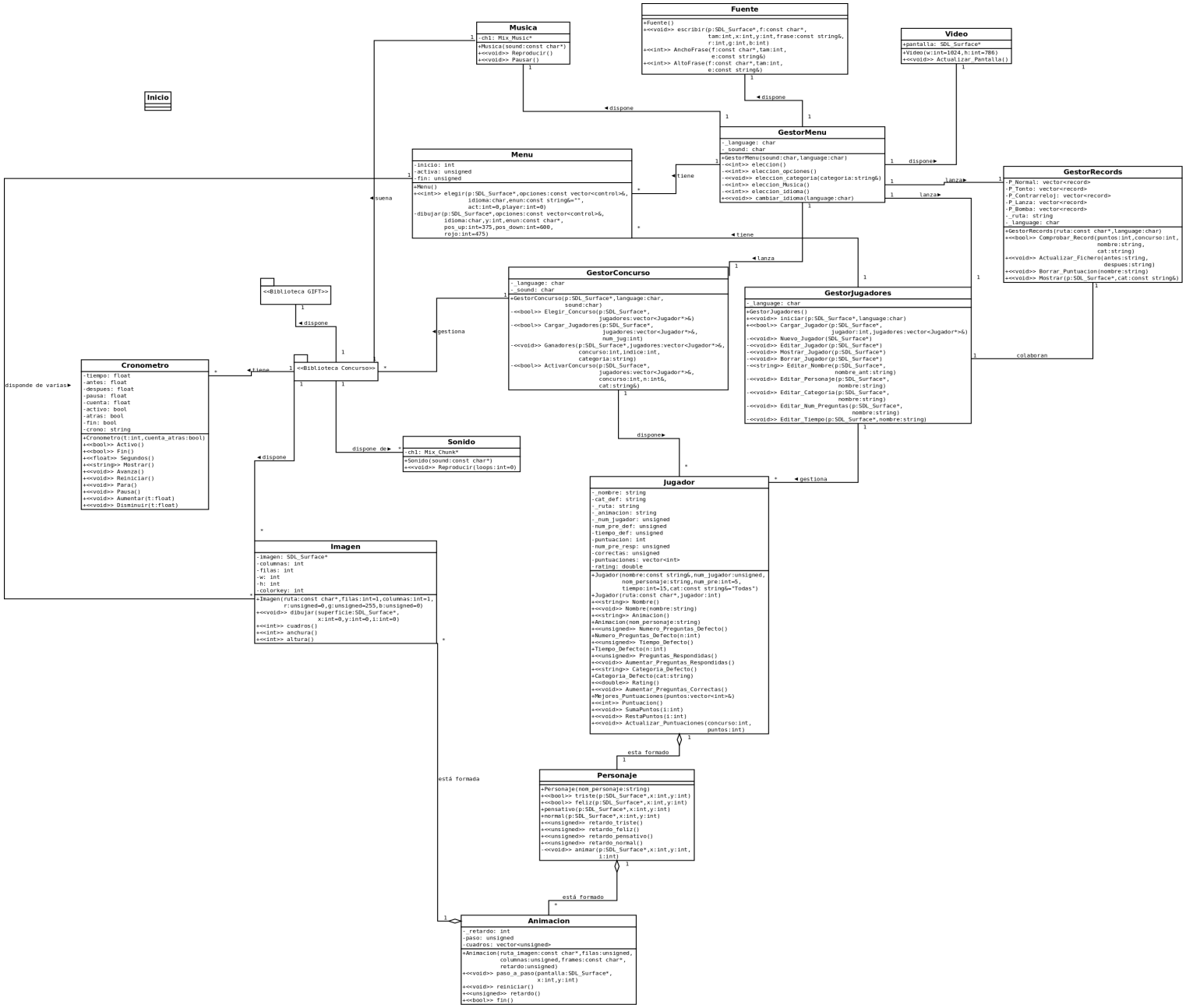

Figura 5.10: Diagrama de Clases de diseño

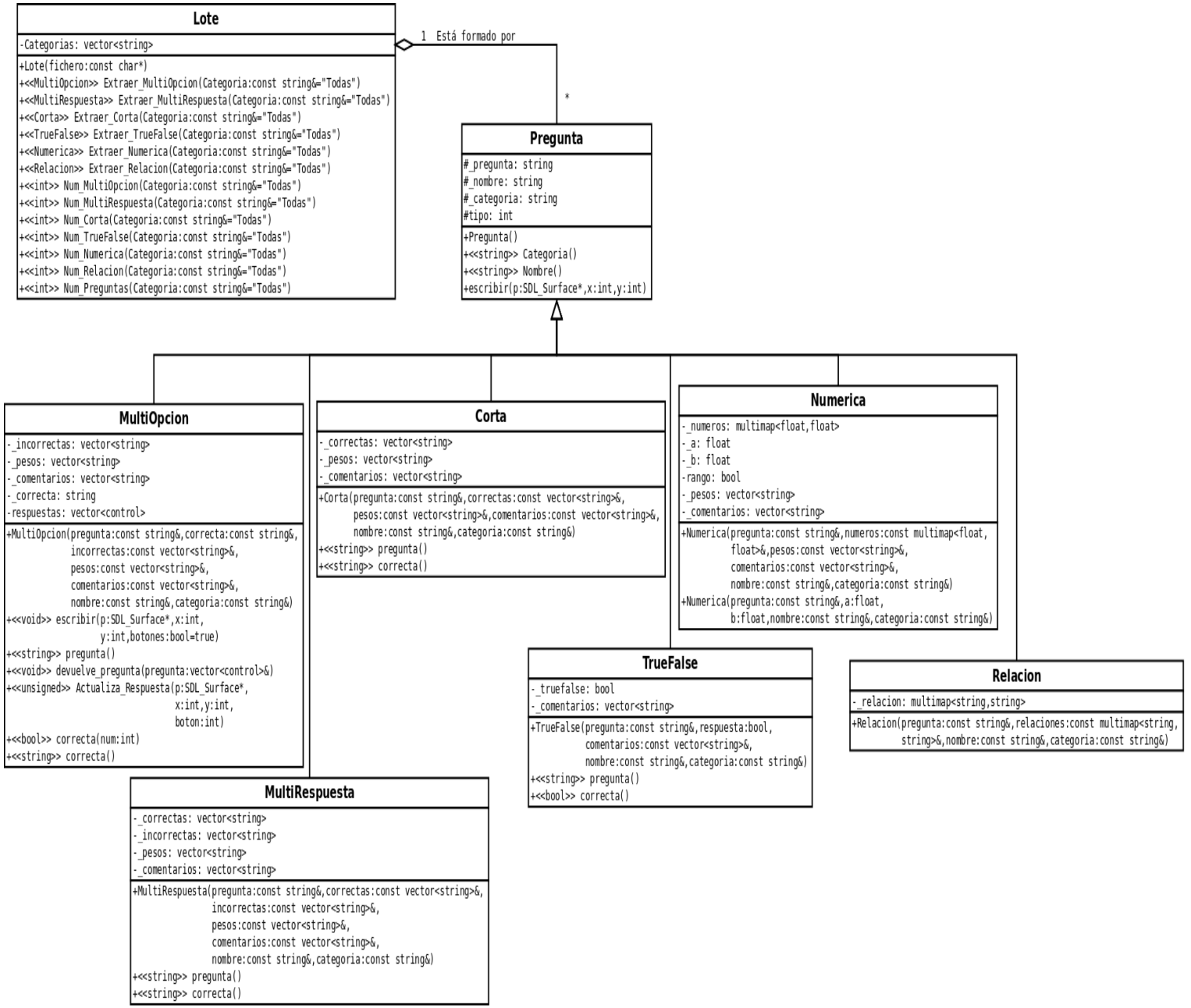

Figura 5.11: Diagrama de Clases Biblioteca GIFT

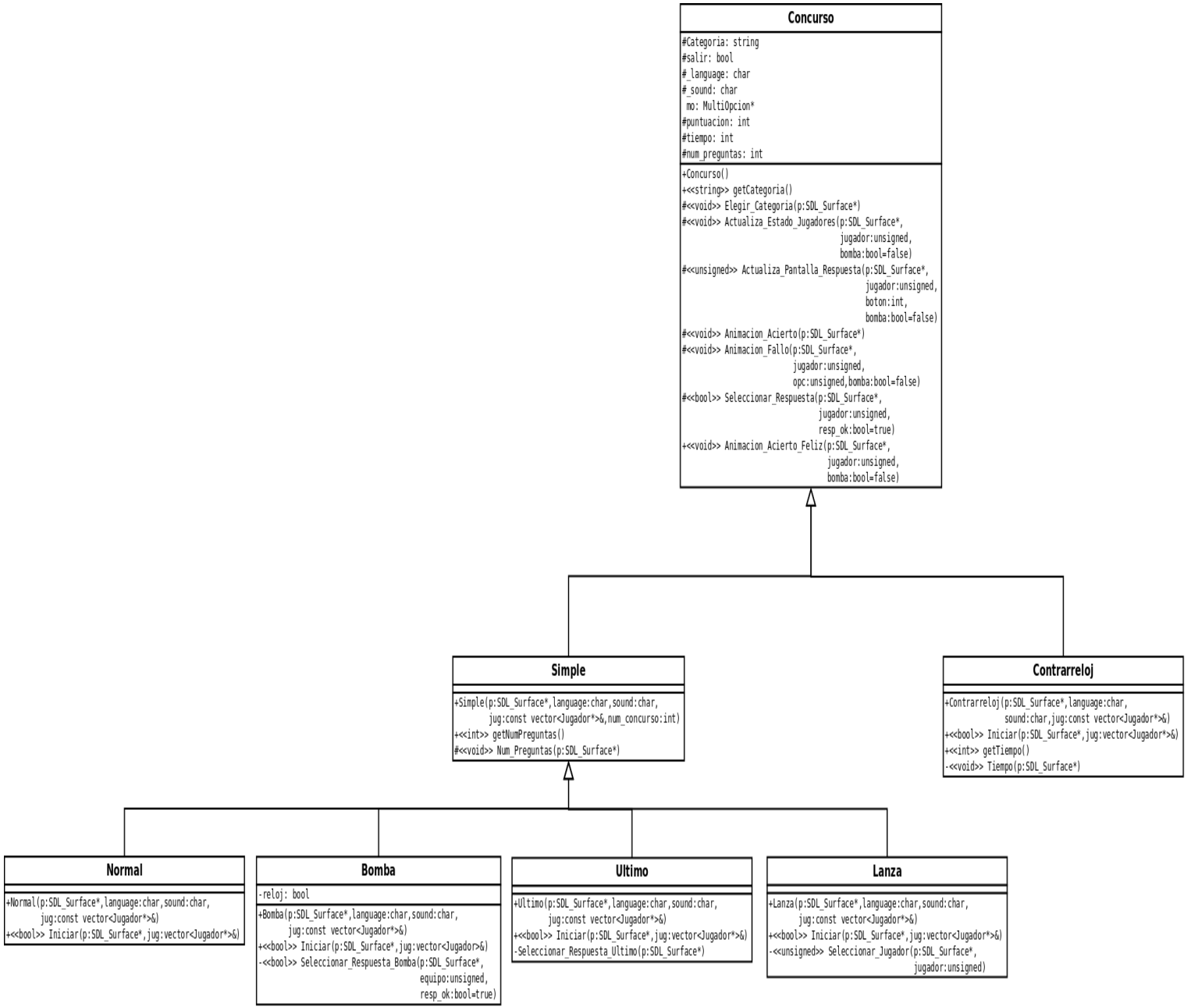

Figura 5.12: Diagrama de Clases Biblioteca Concursos

5.1. GestorMenu

Módulo principal donde se manejan todas las llamadas a las distintas áreas del videojuego.

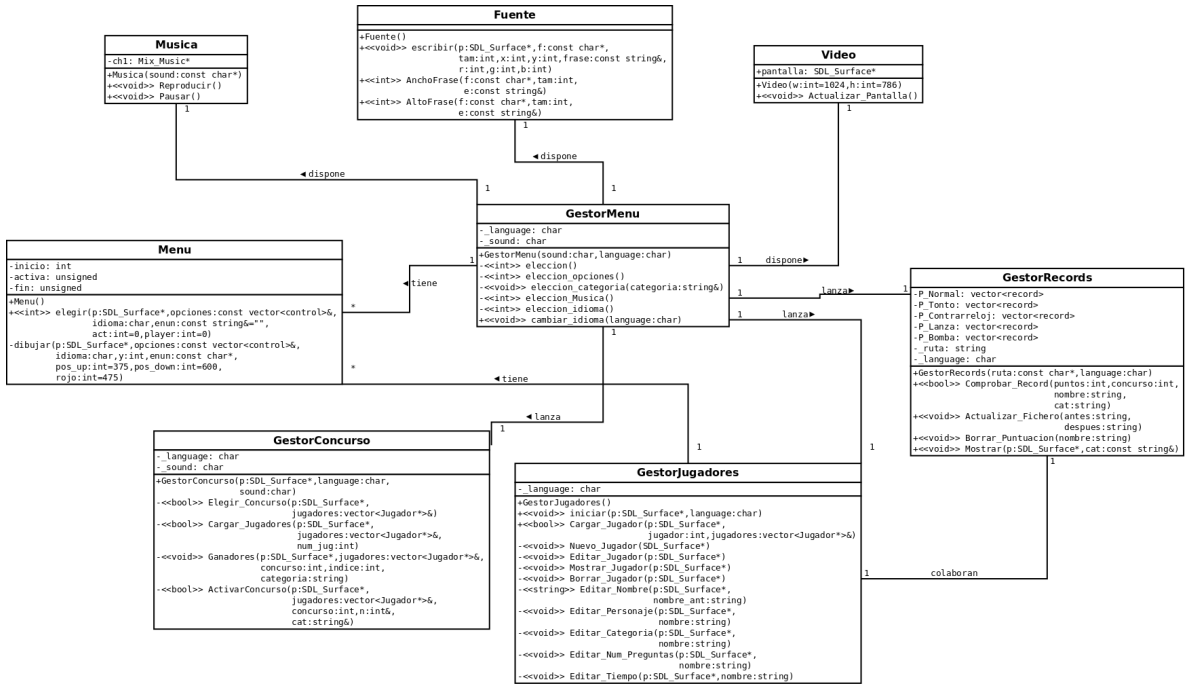

Figura 5.13: Diagrama de clases de la Clase GestorMenu

- **GestorMenu (char Lenguaje, char Sonido)**. Constructor de la Clase GestorMenu. Inicia y se maneja por el menú principal del juego. Es el encargado de modificar, las opciones del juego y de los distintos gestores del videojuego. Se ayuda de funciones privadas para ello. *Lenguaje* indica el lenguaje en que se muestra el menú (Español, Inglés) y *Sonido* indica si el sonido está activo o no.

A continuación se muestra un diagrama con las distintas dependencias del módulo:

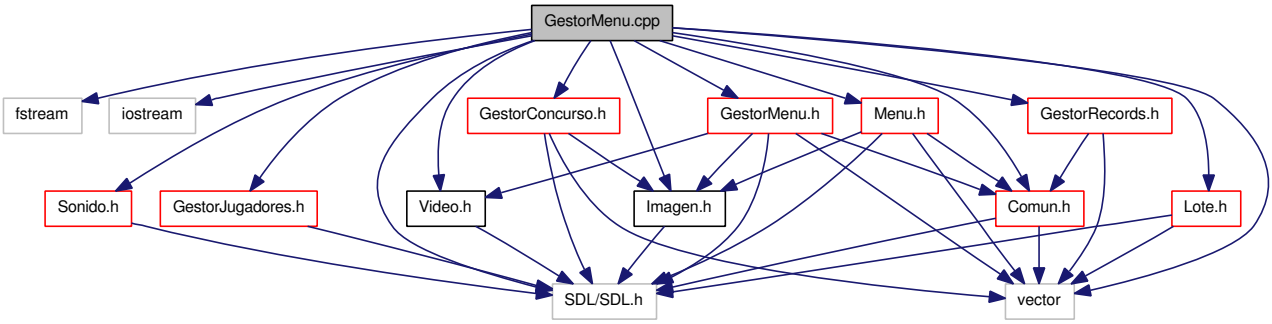

Figura 5.14: Diagrama de dependencia de GestorMenu

5.2. GestorJugadores

Módulo donde se maneja toda la gestión de jugadores.

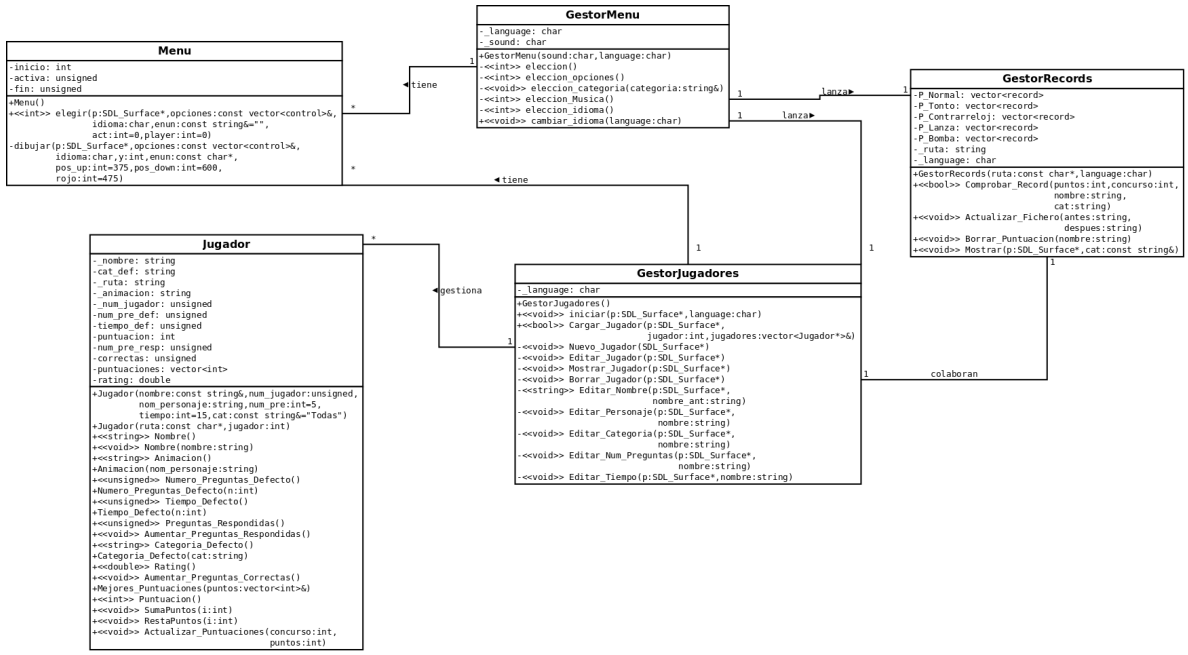

Figura 5.15: Diagrama de clases de la Clase GestorJugadores

- **GestorJugadores (char Lenguaje)**. Constructor de la Clase GestorJugadores. *Lenguaje* indica el lenguaje en que se muestra el menú (Español, Inglés).
- **void iniciar(Pantalla)**. Inicia el menú para la gestión de jugadores. Permite crear, editar, mostrar o borrar jugadores. Se ayuda de funciones privadas para ello.
- **bool Cargar_Jugador(Pantalla, int Jugador, vector<Jugador>Jugadores)**. Carga el jugador número *Jugador* en el vector *Jugadores*. Devuelve true si todo ha ido bien.

A continuación se muestra un diagrama con las distintas dependencias del módulo:

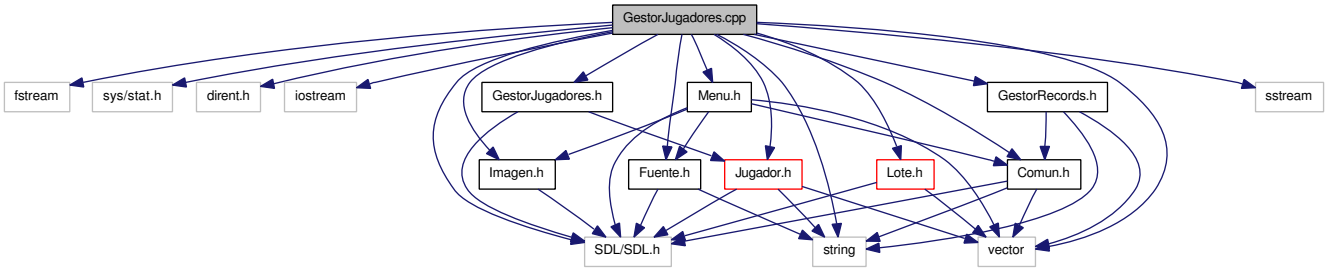

Figura 5.16: Diagrama de dependencia de GestorJugadores

Seguidamente se muestran los diagramas de secuencia asociados al caso de uso Gestor Jugadores:

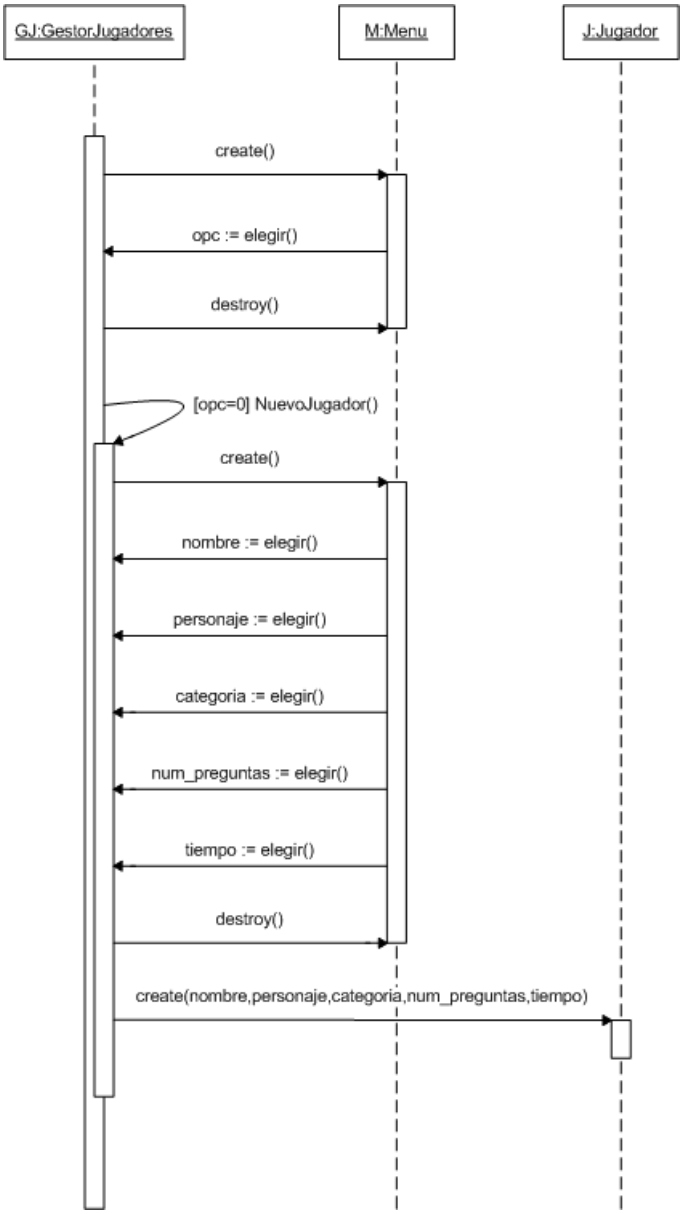

Figura 5.17: Primer Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

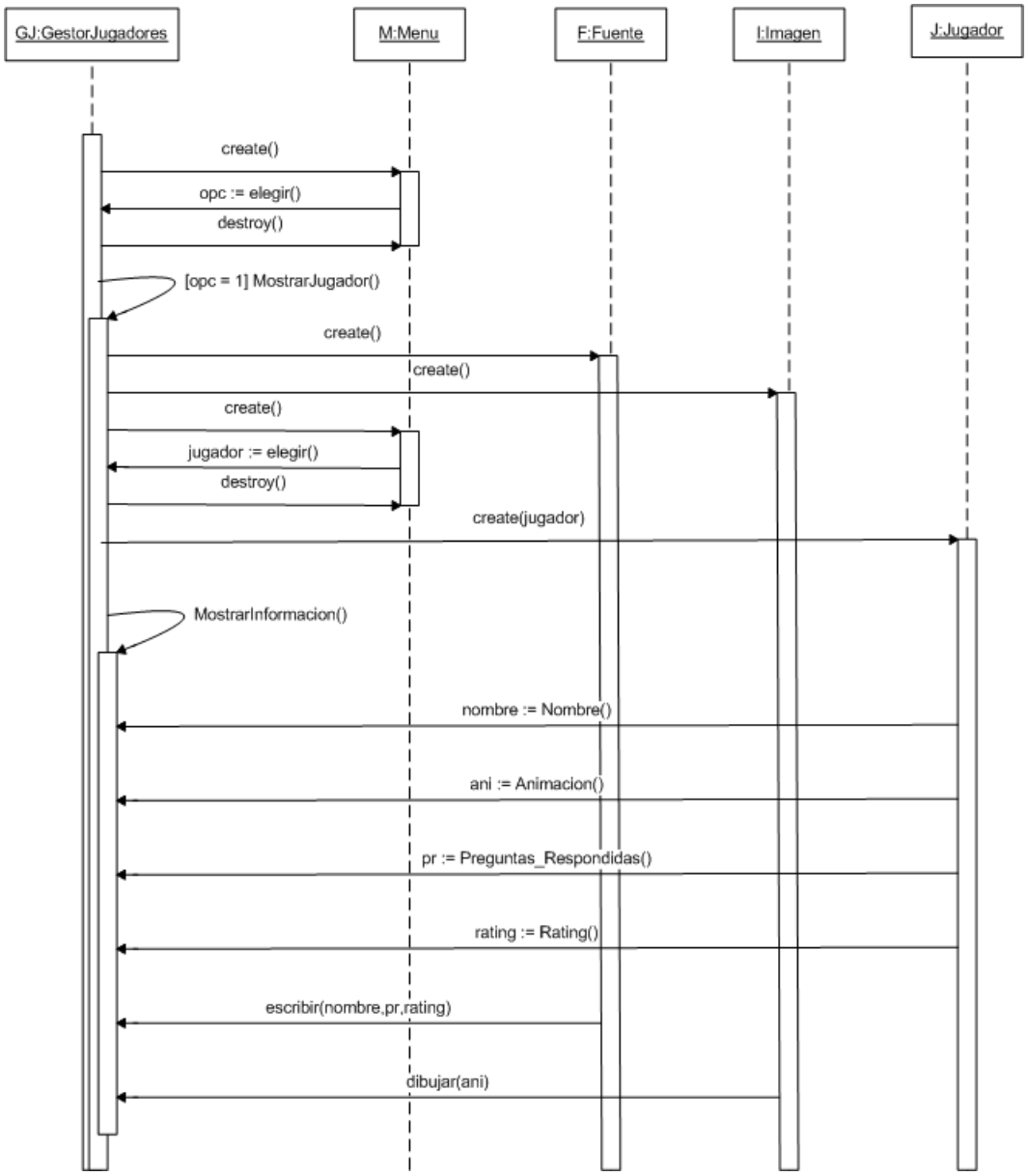

Figura 5.18: Segundo Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

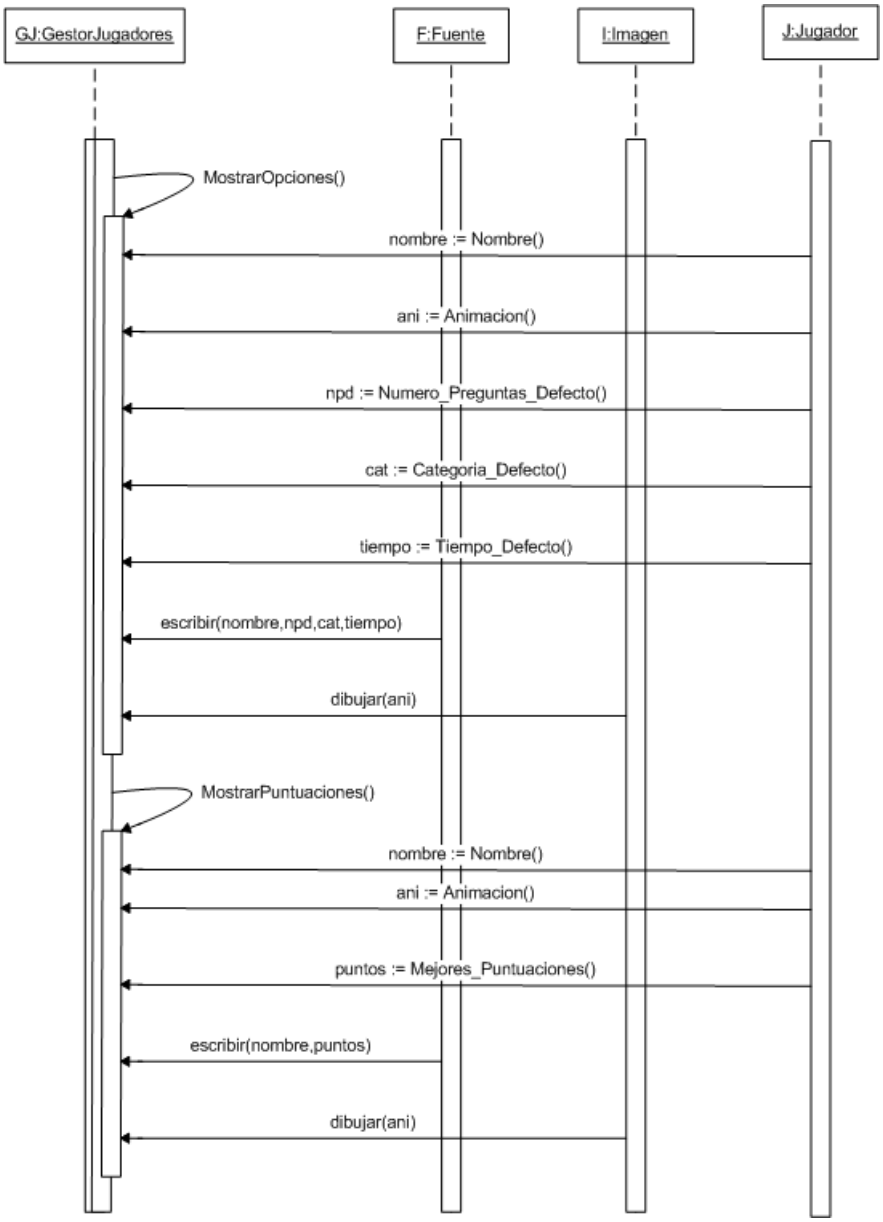

Figura 5.19: Tercer Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

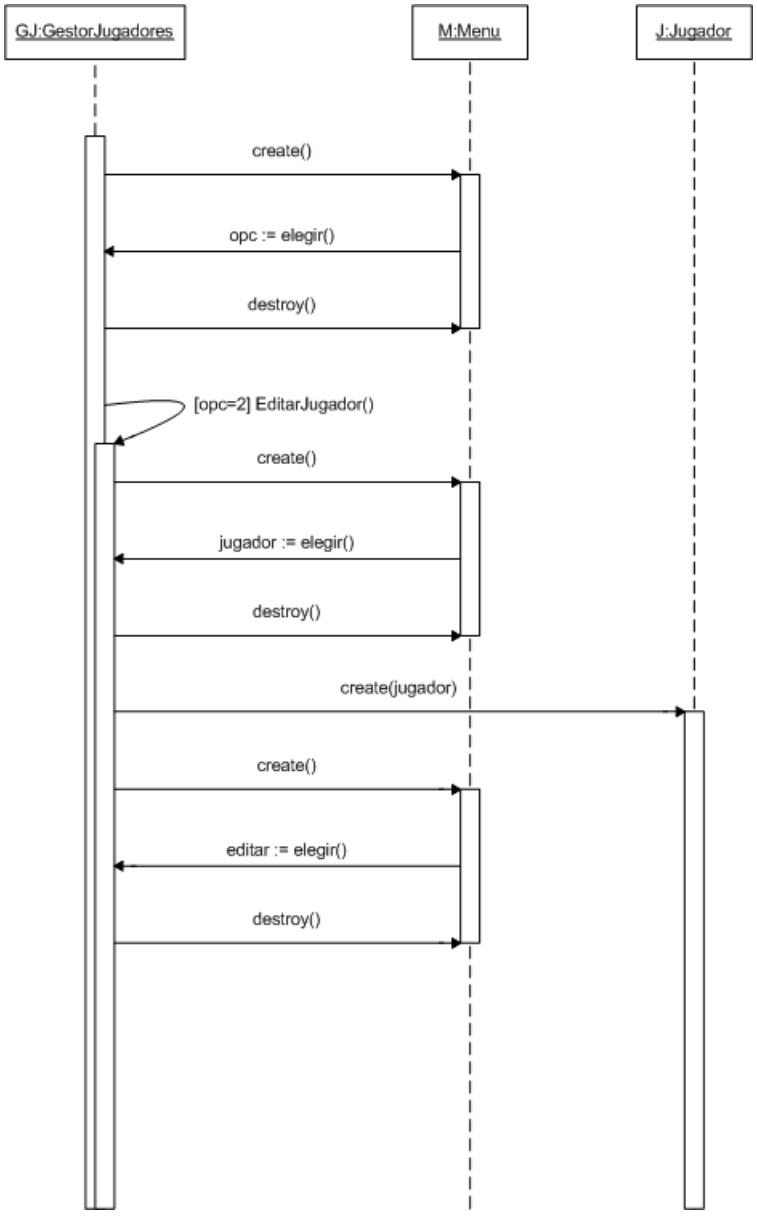

Figura 5.20: Cuarto Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

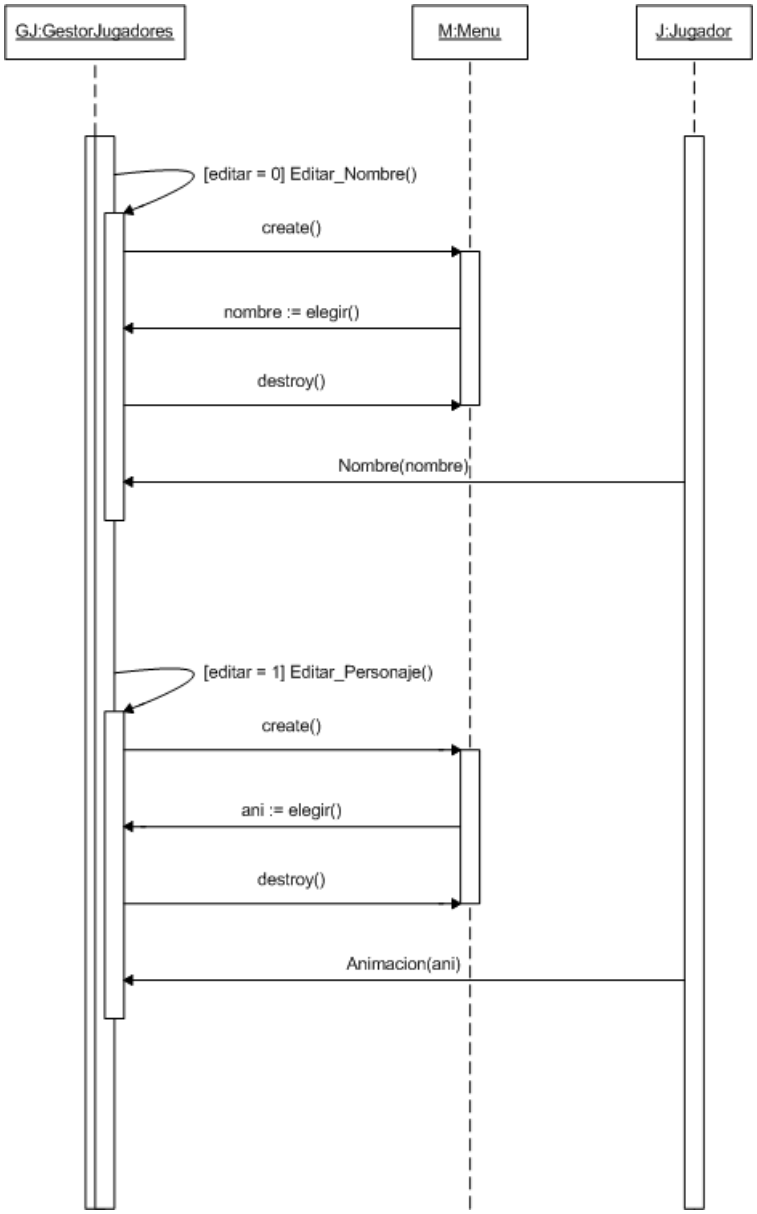

Figura 5.21: Quinto Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

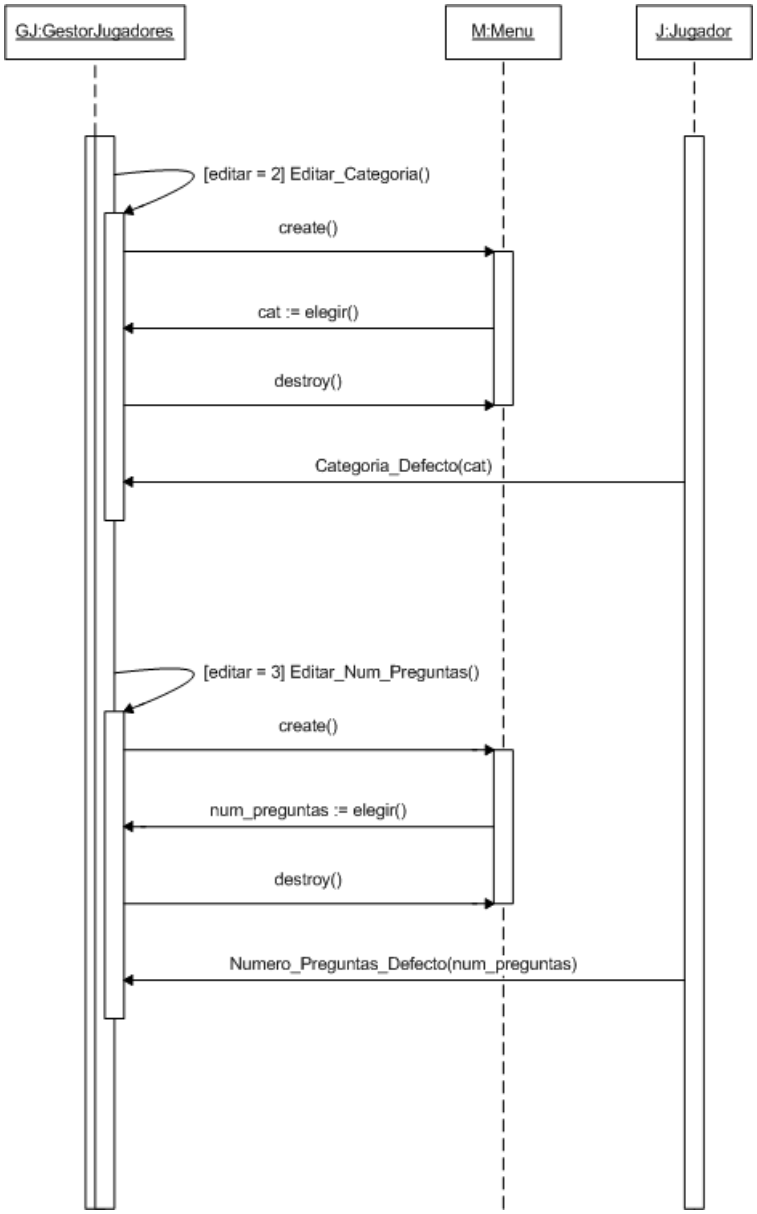

Figura 5.22: Sexto Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

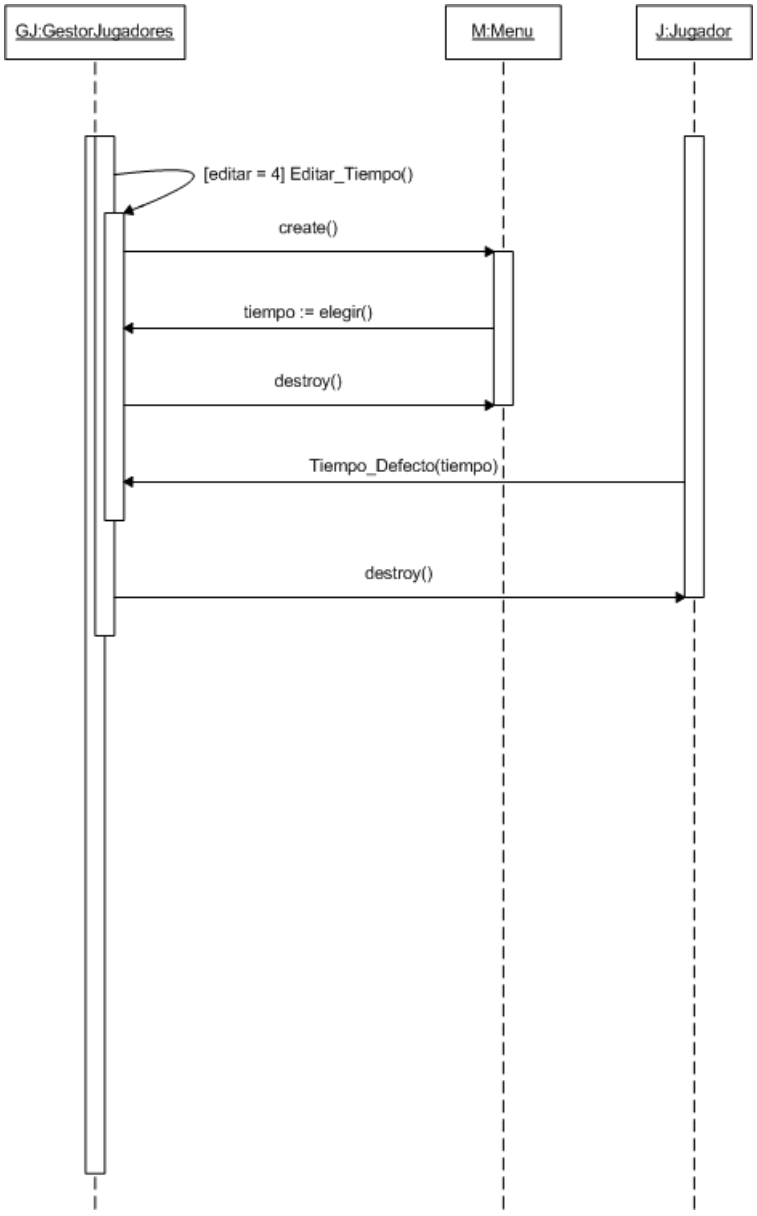

Figura 5.23: Séptimo Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

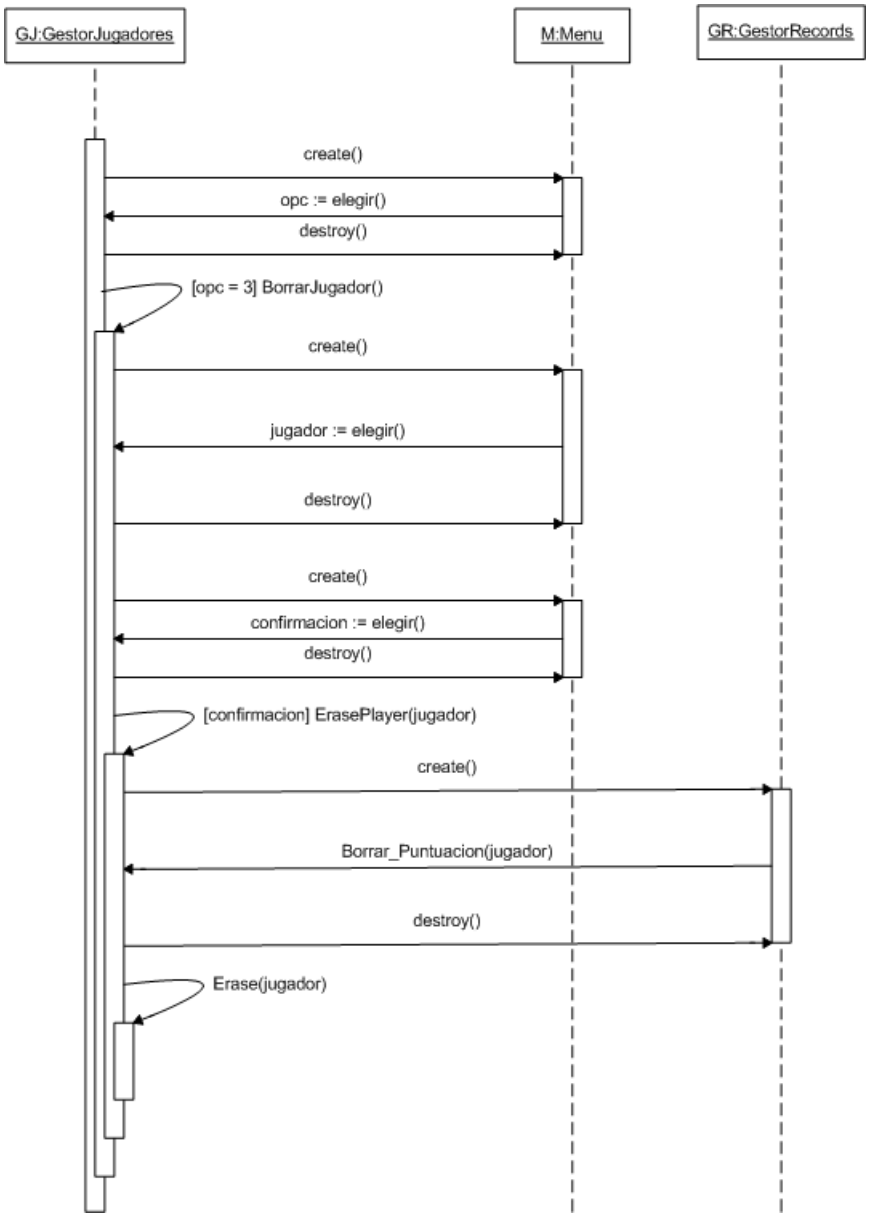

Figura 5.24: Octavo Diagrama de Secuencia asociado al Caso de Uso Gestor Jugadores

5.3. GestorRecords

Módulo donde se maneja las máximas puntuaciones de los concursos.

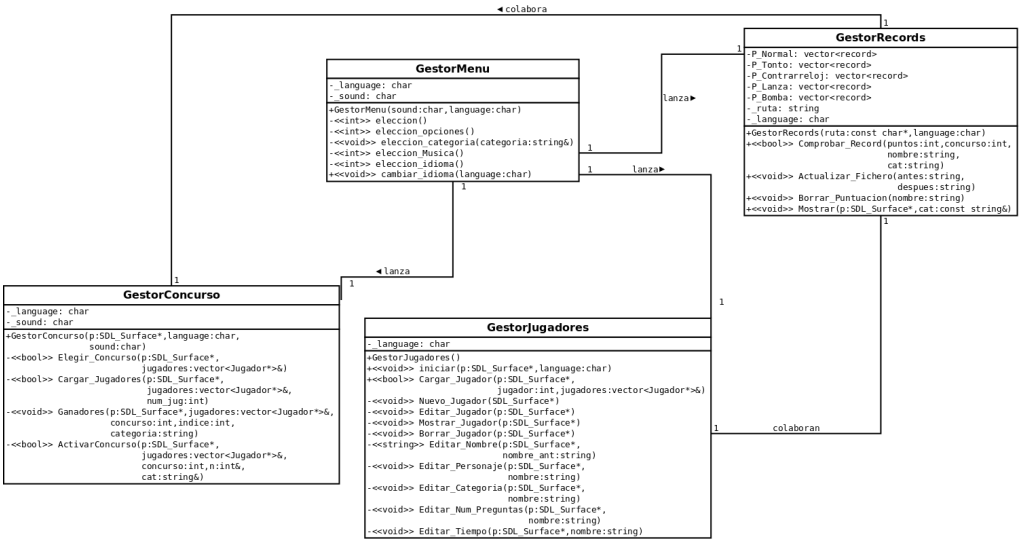

Figura 5.25: Diagrama de clases de la Clase GestorRecords

- **GestorRecords (string Ruta, char Lenguaje)**. Constructor de la Clase GestorRecords. *Ruta* se refiere al fichero donde se guardarán los records y *Lenguaje* indica el lenguaje en que se muestra el menú (Español, Inglés).

- **bool Comprobar_Record(int Puntos, int Concurso, string Nombre, string Categoria)**. Comprueba si el jugador *Jugador* habiendo conseguido *Puntos* puntos en el concurso *Concurso* con categoría *Categoria* ha conseguido un nuevo record. Devuelve true si es así y false en caso contrario.

- **void Actualizar_Fichero(string NombreAntes, string NombreDespues)**. Actualiza el nombre del jugador *NombreAntes* con el nombre *NombreDespues*.

- **void Borrar_Puntuacion(string Nombre)**. Borra la puntuación del jugador *Nombre*.

- **void Mostrar(Pantalla, string Categoria)**. Muestra por *Pantalla* las puntuaciones en la categoría *Categoria* de cada concurso.

A continuación se muestra un diagrama con las distintas dependencias del módulo:

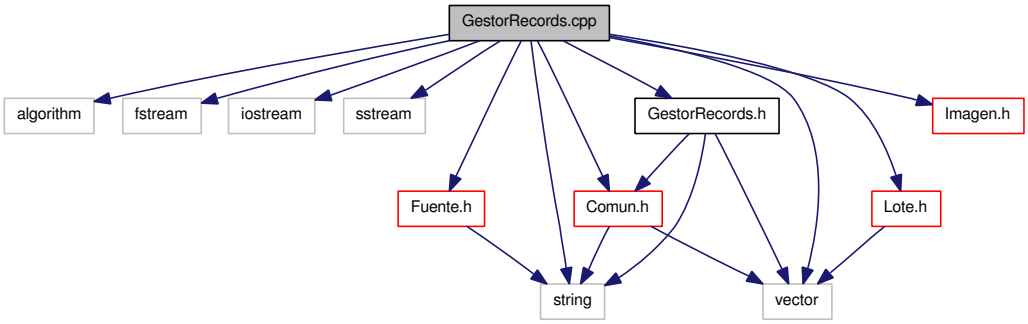

Figura 5.26: Diagrama de dependencia de GestorRecords

5.4. GestorConcurso

Módulo donde se lanzan y controlan los distintos concursos disponibles en el videojuego.

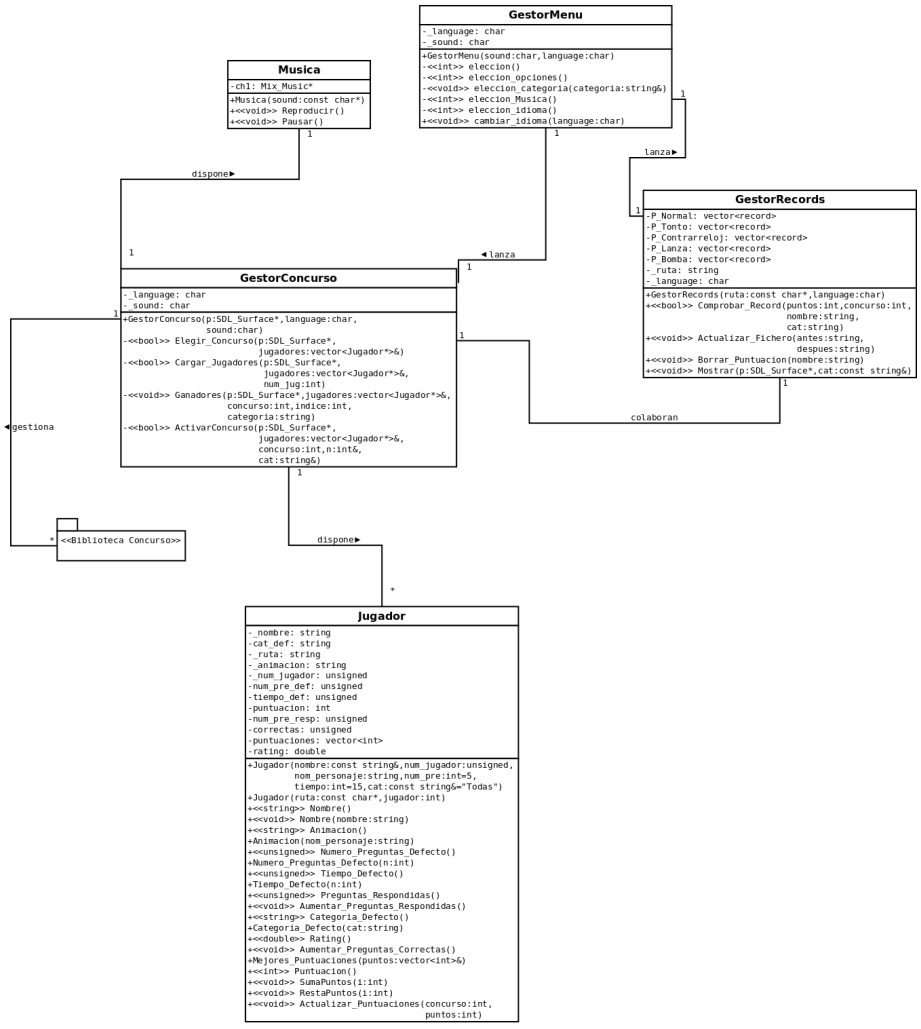

Figura 5.27: Diagrama de clases de la Clase GestorConcurso

- **GestorConcurso(Pantalla, char Lenguaje, char Sonido)**. Constructor de la Clase GestorConcurso. Se encarga de gestionar, lanzar y controlar los distintos concursos que el jugador quiera jugar. Se ayuda de funciones privadas para ello. *Lenguaje* indica el lenguaje en que se muestra el menú (Español, Inglés) y *Sonido* indica si el sonido está activo o no.

A continuación se muestra un diagrama con las distintas dependencias del módulo:

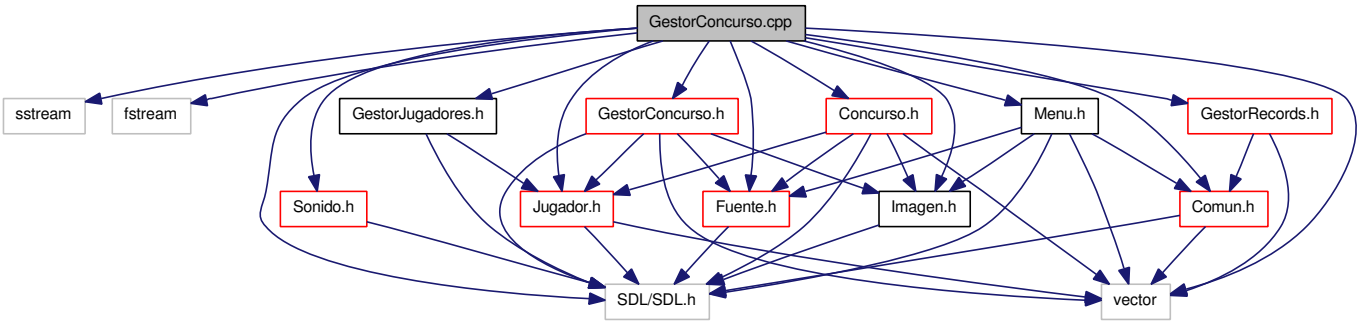

Figura 5.28: Diagrama de dependencia de GestorConcurso

Seguidamente se muestran los diagramas de secuencia asociados al caso de uso Jugar a un Concurso:

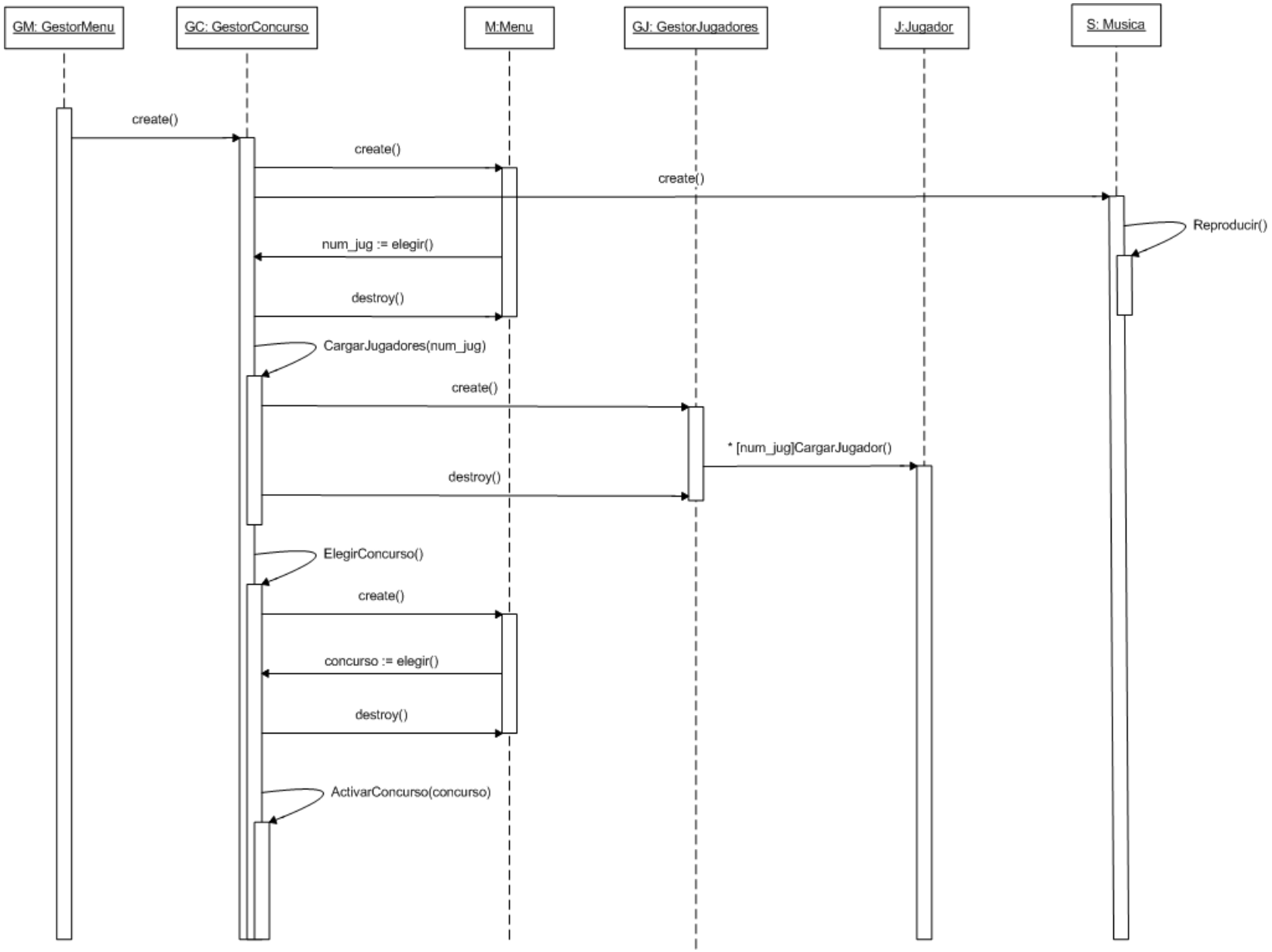

Figura 5.29: Primer Diagrama de Secuencia asociado al Caso de Uso: Jugar a un Concurso

Figura 5.30: Segundo Diagrama de Secuencia asociado al Caso de Uso: Jugar a un Concurso

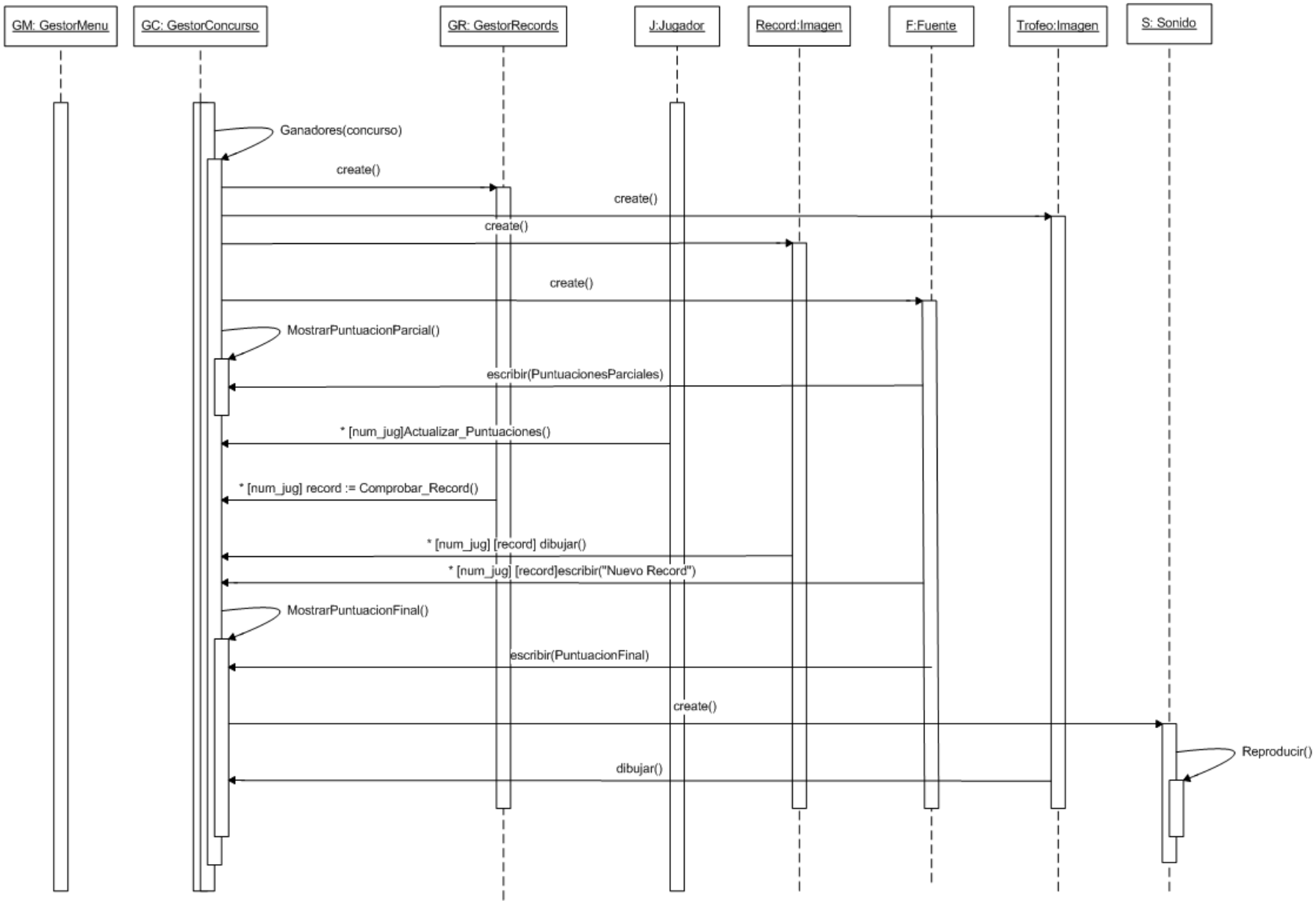

Figura 5.31: Tercer Diagrama de Secuencia asociado al Caso de Uso: Jugar a un Concurso

5.5. Lote

Módulo para manejar las preguntas.

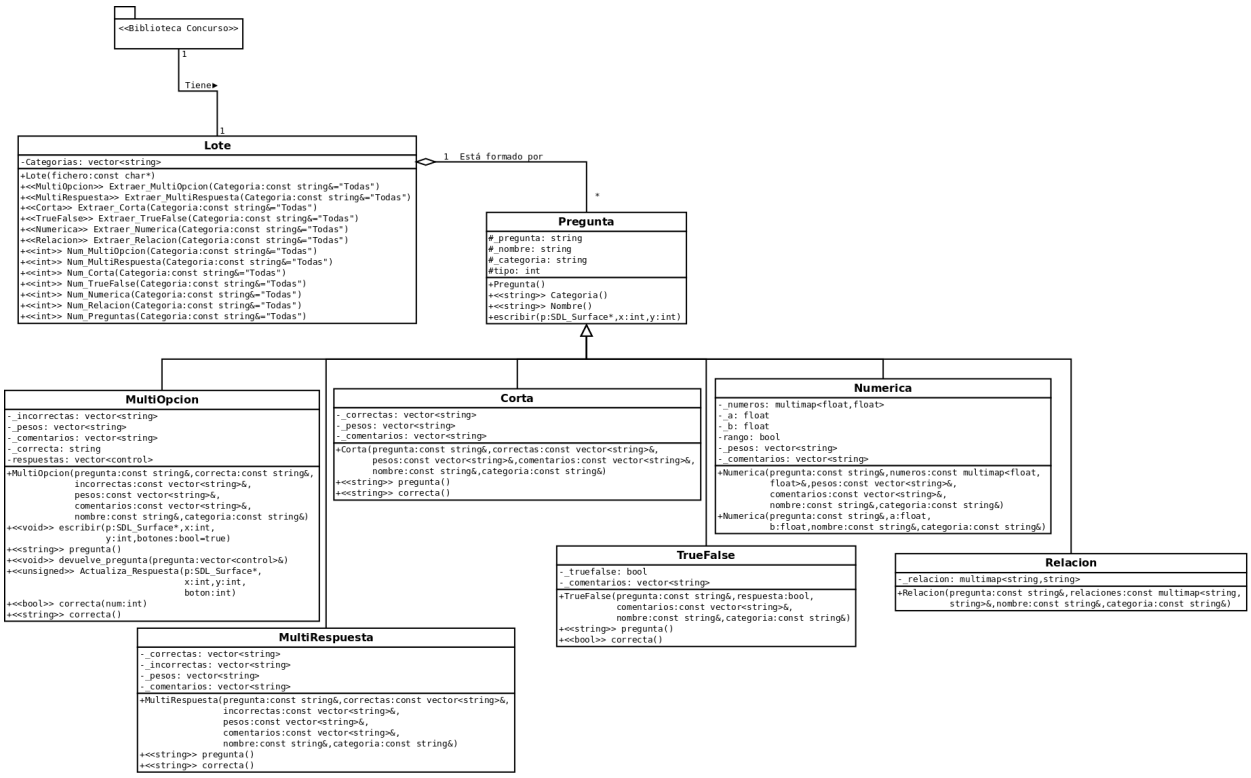

Figura 5.32: Diagrama de clases de la Clase Lote

- **Lote(string Fichero)**. Constructor que usaremos para inicializar la biblioteca de preguntas, a la que le pasamos el fichero Preguntas.gift para que lea las preguntas almacenadas en dicho fichero. Después de construir el objeto, todas las preguntas estarán en memoría principal.
- **MultiOpcion Extraer_MultiOpcion(string Categoria)**. Devuelve una pregunta del tipo MultiOpcion. En Categoría se le puede indicar la categoría de la pregunta a extraer. Debe asegurarse que existen preguntas con la función Num_MultiOpcion() explicada a continuación.
- **int Num_MultiOpcion(string Categoria)**. Devuelve el número de preguntas MultiOpcion que hay en el Lote. Como en la función anterior se puede indicar la categoría para saber cuantas hay de dicha categoría. Hay dos funciones como las anteriores por cada tipo de pregunta. Tienen el mismo funcionamiento, unicamente devuelven tipos de preguntas distintos. Por tanto, se obvian para que no se haga pesado.
- **void Todas_Categorias(vector<string>Categorias)**. Devuelve en el vector que se le pasa por parámetro, todas las categorías disponibles en el Lote.
- **int Num_Preguntas(string Categoria)**. Devuelve el número de preguntas totales que hay en el Lote. También se puede filtrar las preguntas por categoría indicandolo en el parámetro.

Como en la sección anterior vemos el diagrama de dependencias del módulo:

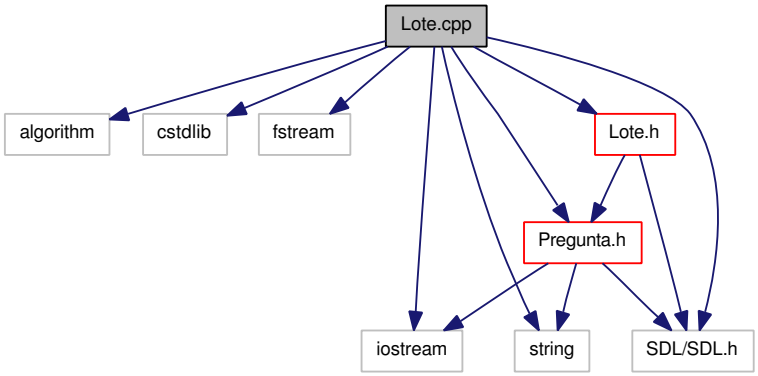

Figura 5.33: Diagrama de dependencia de Lote

5.6. Pregunta

Módulo que representa las diferentes preguntas.

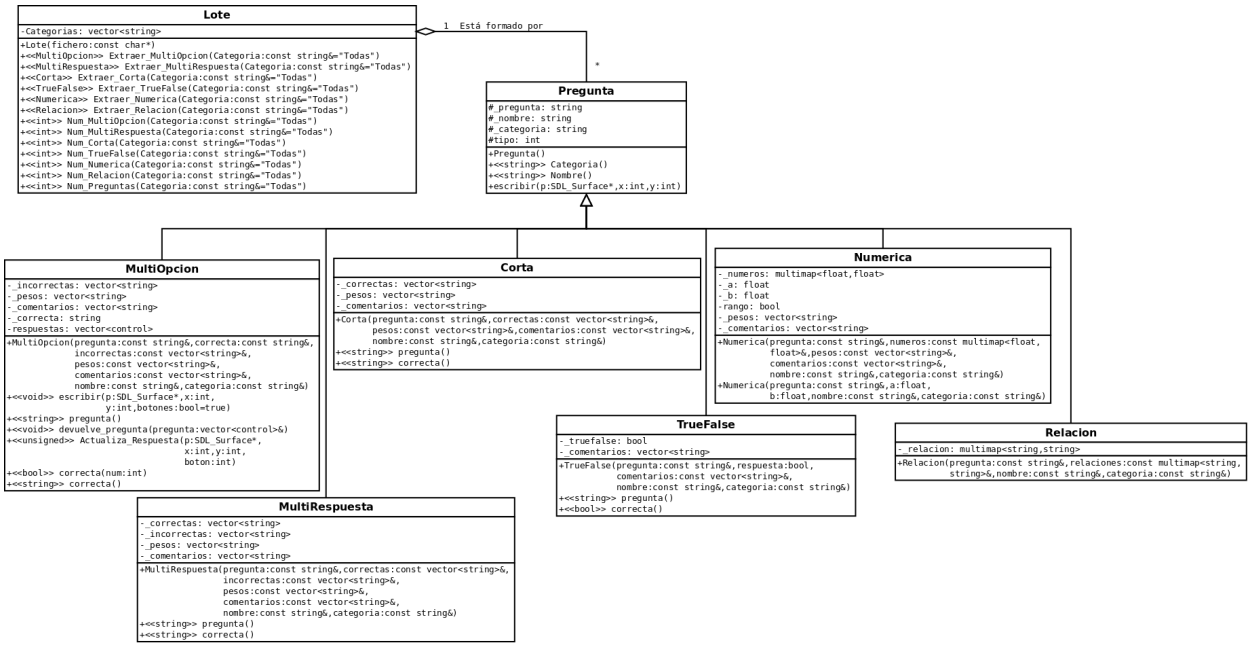

Figura 5.34: Diagrama de clases de la Clase Pregunta

- **Pregunta**(). Crea una pregunta sin tipo asignado.
- **string Categoria**(). Devuelve la categoría de la pregunta.
- **string Nombre**(). Devuelve el nombre de la pregunta.

A continuación se muestra el diagrama de dependencias del módulo:

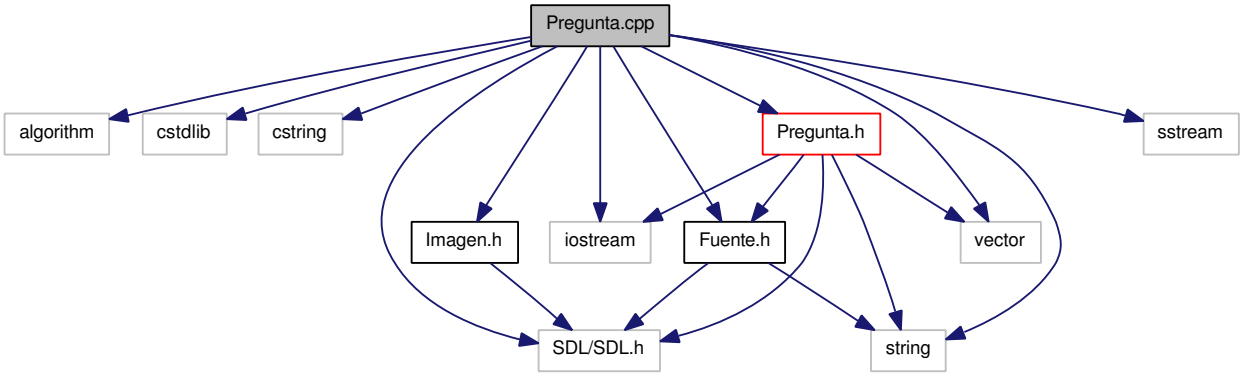

Figura 5.35: Diagrama de dependencia de Pregunta

Esta clase está especializada en los diferentes tipos de preguntas que se presentan a continuación.

5.6.1. MultiOpcion

Esta especialización se encarga de manejar las preguntas de tipo MultiOpcion[1].

- **MultiOpcion(string Pregunta,string Correcta, vector<string>Incorrectas, vector<string>Pesos, vector<string>Comentarios, string Nombre, string Categoria)**. Crea una pregunta del tipo MultiOpcion. El parámetro *Pregunta* indica la pregunta en modo texto, el parámetro *Correcta* indica la respuesta correcta en modo texto, el parámetro *Incorrectas* indica las respuestas incorrectas, el parámetro *Pesos* indica los pesos de cada respuesta, el parámetro *Comentarios* indica los comentarios de cada respuesta, el parámetro *Nombre* se refiere al nombre de la pregunta y el parámetro *Categoría* hace referencia a la categoría de la pregunta.
- **string pregunta()**. Devuelve el texto de la pregunta.
- **bool correcta(int Num)**. Devuelve si la respuesta número *Num* es la respuesta correcta.
- **string correcta()**. Devuelve el texto de la respuesta correcta.

[1]Unicamente se va a exponer la clase MultiOpcion, las demás tienen las mismas funciones que sirven para las mismas cosas. Así que para que no se haga tan pesado se toma como ejemplo la clase MultiOpcion.

5.7. Menu

Módulo para crear y administrar menú.

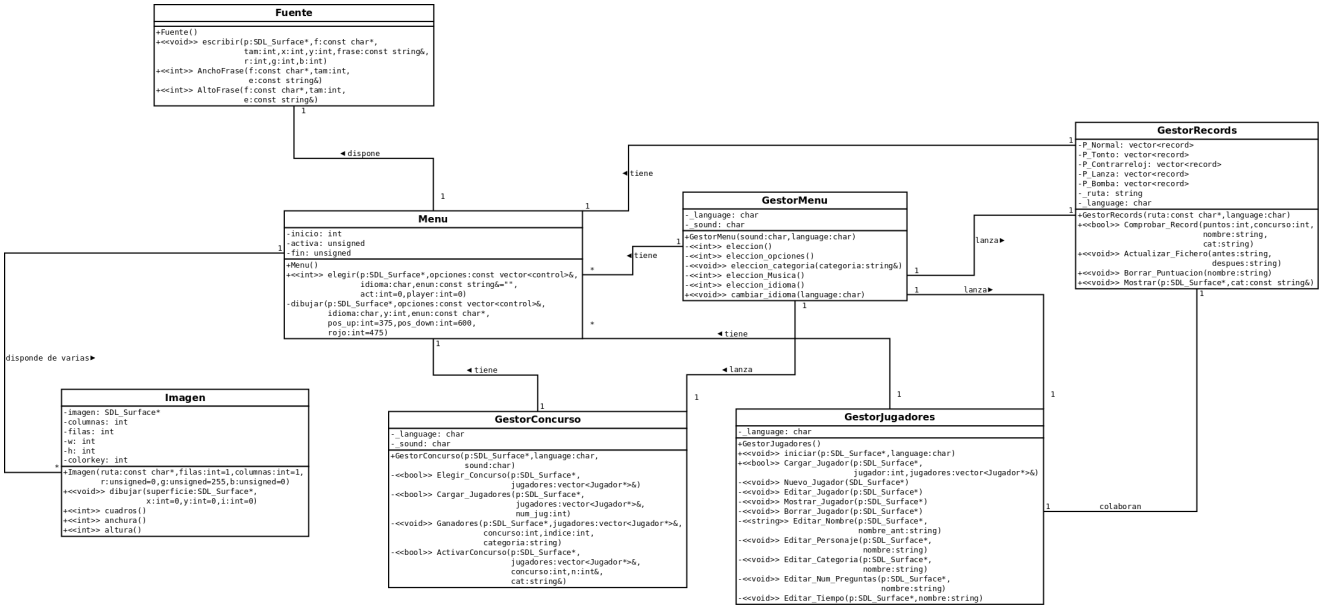

Figura 5.36: Diagrama de clases de la Clase Menu

- **Menu**(). El constructor de la clase Menu.
- **int elegir(Pantalla)**. Función para elegir una opción del menú creado. Devuelve el número de la opción.

A continuación, se muestra el diagrama de dependencias del módulo:

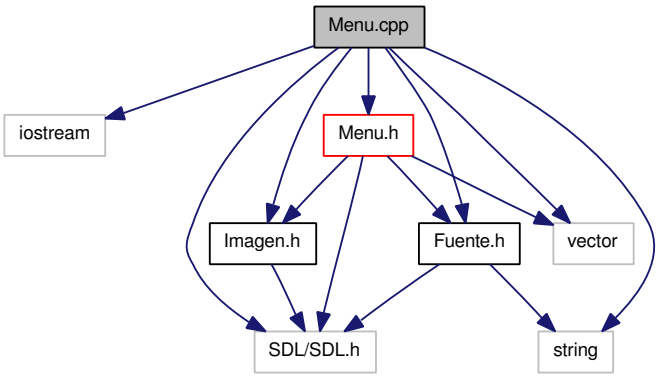

Figura 5.37: Diagrama de dependencia de Menu

5.8. Concurso

Módulo que ejecuta y controla los distintos concursos del juego.

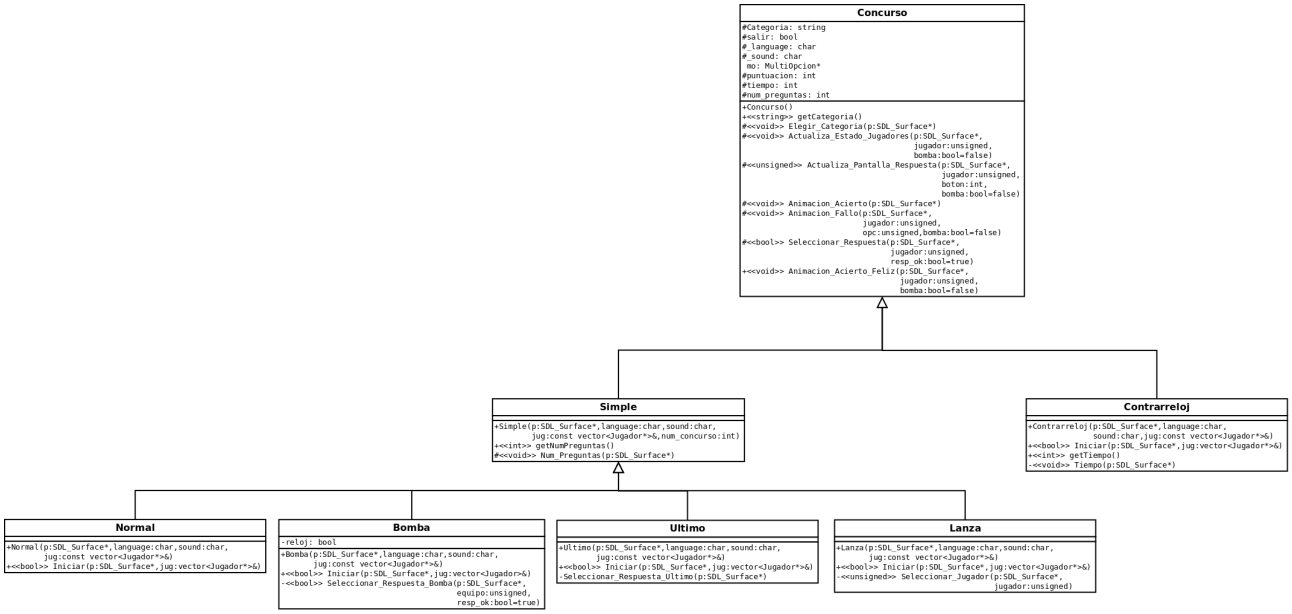

Figura 5.38: Diagrama de clases de la Clase Concurso

- **Concurso::Concurso()** El constructor de la clase Concurso. Inicializa todos los datos compartidos por los concursos.

A continuación se muestra el diagrama de dependencias del módulo:

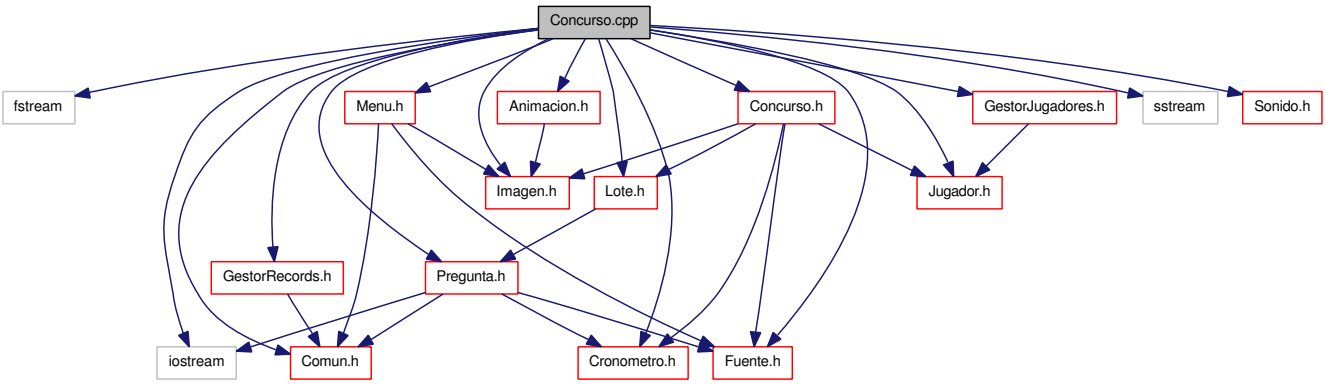

Figura 5.39: Diagrama de dependencia de Concurso

5.8.1. Simple

Especialización de la clase Concurso para controlar las concurso simples.

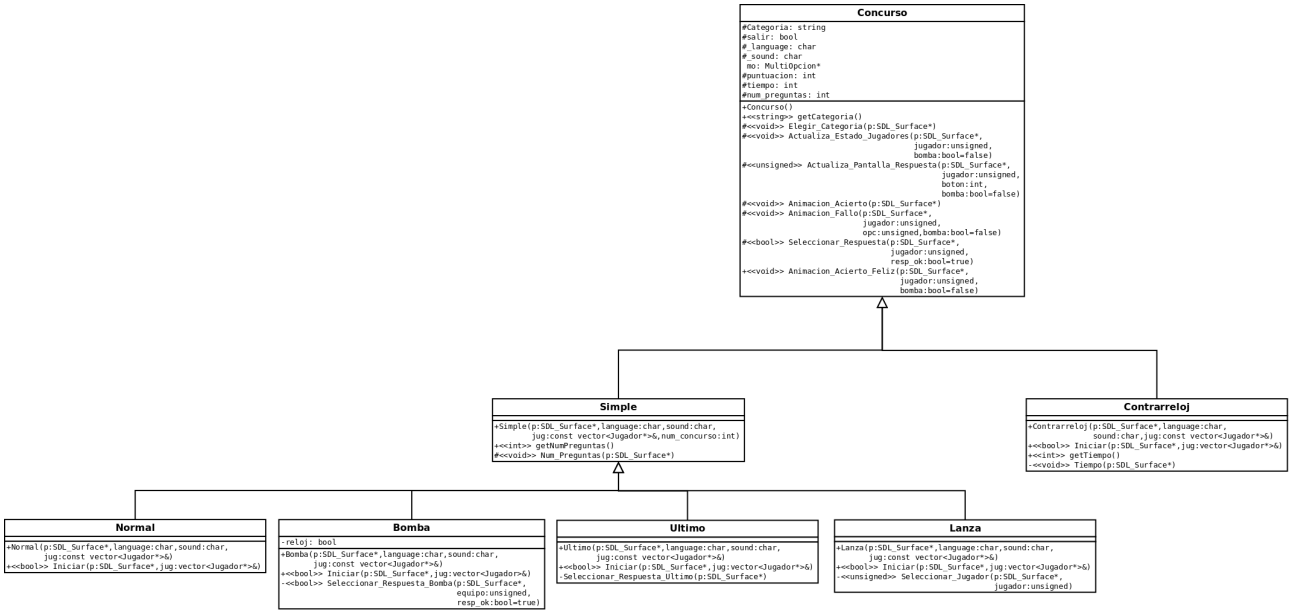

Figura 5.40: Diagrama de clases de la Clase Simple

- **Simple(Pantalla, char Lenguaje, char Sonido, vector<Jugador>Jugadores, int NumConcurso)**. El constructor de la clase Simple. Inicializa todos los datos compartidos por los concursos simples. Recoge los jugadores de *Jugadores* y recibe el identificador de concurso *NumConcurso*.
- **int getNumPreguntas()**. Devuelve el número de preguntas a las que está configurado el concurso.

Normal

Especialización de la clase Simple para controlar el concurso **Normal**.

- **Normal(Pantalla, char Lenguaje, char Sonido, vector<Jugador>Jugadores)**. El constructor de la clase Normal. Inicializa los datos para el concurso **Normal**.
- **Iniciar(Pantalla, vector<Jugador>Jugadores)**. Inicia el concurso **Normal** y lo gestiona hasta su finalización. Devuelve en Jugadores el estado de estos al finalizar el concurso.

Bomba

Especialización de la clase Simple para controlar el concurso **Pasa la Bomba**.

- **Bomba(Pantalla, char Lenguaje, char Sonido, vector<Jugador>Jugadores)**. El constructor de la clase Bomba. Inicializa los datos para el concurso **Pasa la Bomba**.
- **Iniciar(Pantalla, vector<Jugador>Jugadores)**. Inicia el concurso **Pasa la Bomba** y lo gestiona hasta su finalización. Devuelve en Jugadores el estado de estos al finalizar el concurso.

Ultimo

Especialización de la clase Simple para controlar el concurso **Tonto el Último**.

- **Ultimo(Pantalla, char Lenguaje, char Sonido, vector<Jugador>Jugadores)**. El constructor de la clase Ultimo. Inicializa los datos para el concurso **Tonto el Último**.

- **Iniciar(Pantalla, vector<Jugador>Jugadores)**. Inicia el concurso **Tonto el Último** y lo gestiona hasta su finalización. Devuelve en Jugadores el estado de estos al finalizar el concurso.

Lanza

Especialización de la clase Simple para controlar el concurso **Lanza la pregunta**.

- **Lanza(Pantalla, char Lenguaje, char Sonido, vector<Jugador>Jugadores)**. El constructor de la clase Lanza. Inicializa los datos para el concurso **Lanza la Pregunta**.

- **Iniciar(Pantalla, vector<Jugador>Jugadores)**. Inicia el concurso **Lanza la Pregunta** y lo gestiona hasta su finalización. Devuelve en Jugadores el estado de estos al finalizar el concurso.

5.8.2. Contrarreloj

Especialización de la clase Concurso para controlar el concurso **Contrarreloj**.

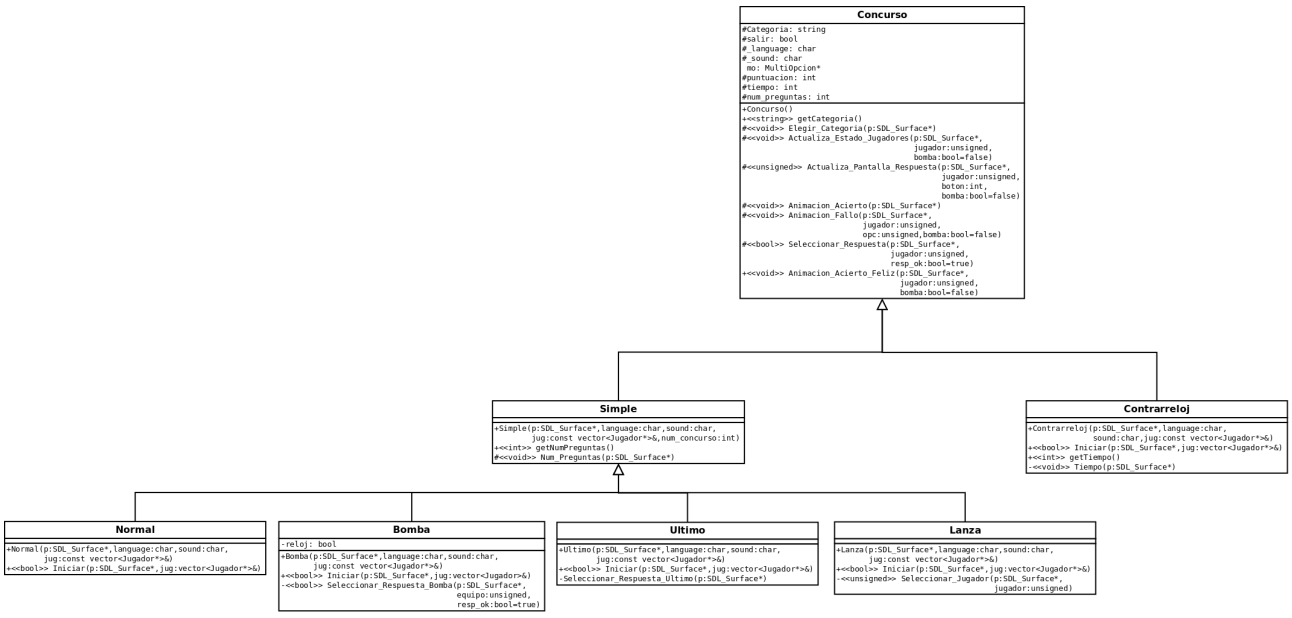

Figura 5.41: Diagrama de clases de la Clase Contrarreloj

- **Contrarreloj(Pantalla, char Lenguaje, char Sonido, vector<Jugador>Jugadores, int NumConcurso)**. El constructor de la clase Contrarreloj. Inicializa todos los datos compartidos por los concursos simples. Recoge los jugadores de *Jugadores* y recibe el identificador de concurso *NumConcurso*.

- **int getTiempo()**. Devuelve el tiempo de contrarreloja al que está configurado el concurso.

5.9. Cronometro

Módulo encargado del cronometro del videojuego.

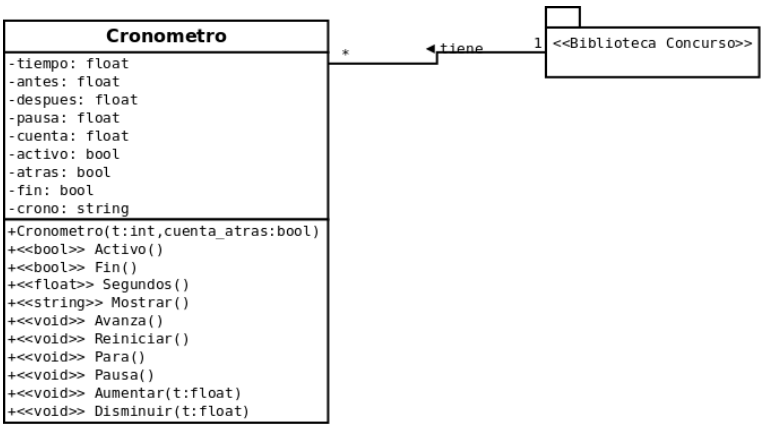

Figura 5.42: Diagrama de clases de la Clase Cronometro

- **Cronometro(int Tiempo, bool Atras)**. El constructor de la clase Cronometro. Crea un cronómetro de tiempo *Tiempo*. Si el parámetro textitAtras es true, indicará que es cuenta atras.
- **bool Fin()**. Indica si el cronometro ha llegado al final.
- **string Mostrar()**. Muestra el tiempo de cronometro.
- **void Avanza()**. Inicia el cronometro.
- **void Reiniciar()**. Reinicia el cronometro.
- **void Disminuir(float t)**. Disminuye *t* segundos al cronometro.
- **void Aumentar(float t)**. Aumenta *t* segundos al cronometro.
- **void Pausa()**. Pausa el cronometro.
- **void Para()**. Para el cronometro.
- **bool Activo()**. Indica si el cronometro está activado o desactivado.
- **float Segundos()**. Devuelve los segundos que quedan de cronometro.

A continuación se muestra el diagrama de dependencias del módulo:

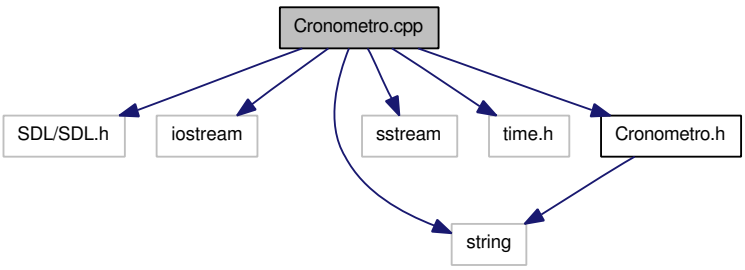

Figura 5.43: Diagrama de dependencia de Cronometro

5.10. Jugador

Módulo encargado de guardar y gestionar los jugadores cuando están en memoria principal.

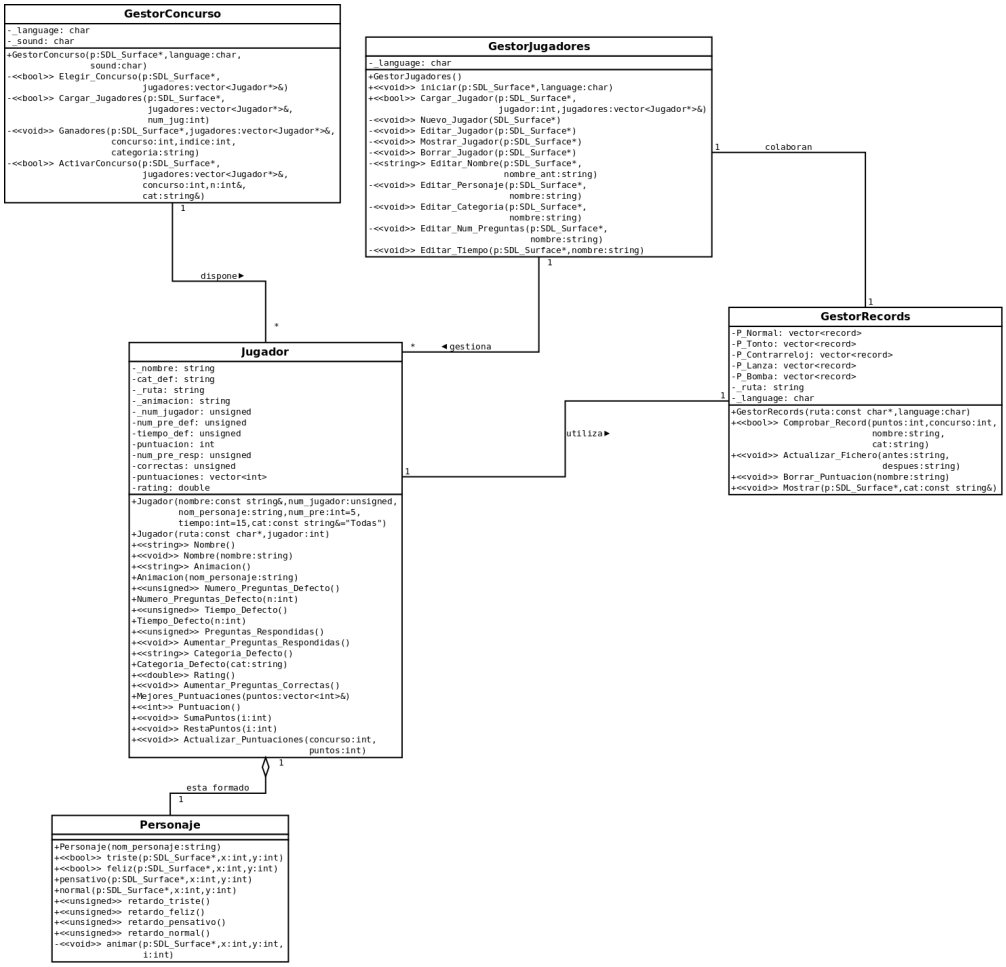

Figura 5.44: Diagrama de clases de la Clase Jugador

- **Jugador(string Nombre, unsigned NumJugador, string NomPersonaje, int NumPreguntas, int Tiempo, string Categoria)**. El constructor de la clase Jugador. Recibe un *Nombre* de jugador, con su *NumJugador*, el nombre del personaje *NomPersonaje*, y las opciones de concurso *NumPreguntas*, *Tiempo* y *Categoria*.
- **Jugador(string Ruta, int NumJugador)**. Este constructor se utiliza para cargar un jugador ya creado. Recibe la ruta *Ruta* en donde se encuentran los datos del jugador y el número de jugador *NumJugador*.
- **Jugador(const Jugador& j)**. Constructor de copia.
- **Jugador& operator =(const Jugador& j)**. Operador de asignación.
- **string Nombre()**. Devuelve el nombre del jugador.
- **void Nombre(string Nombre)**. Cambia el nombre de jugador por *Nombre*.
- **unsigned Numero_Jugador()**. Devuelve el número del jugador.

- **string Animacion()**. Devuelve el nombre del personaje que lo representa.
- **void Animacion(string Nompersonaje)**. Cambia el personaje del jugador por *NomPersonaje*.
- **unsigned Numero_Preguntas_Defecto()**. Devuelve el número de preguntas por defecto del jugador.
- **void Numero_Preguntas_Defecto(int n)**. Cambia el número de preguntas por defecto del jugador por *n*.
- **int Tiempo_Defecto()**. Devuelve el tiempo de contrarreloj por defecto del jugador.
- **void Tiempo_Defecto(int n)**. Cambia el tiempo de contrarreloj por defecto del jugador por *n*.
- **string Categoria_Defecto()**. Devuelve la categoría de las preguntas por defecto del jugador.
- **Categoria_Defecto(string cat)**. Cambia la categoría de las preguntas por defecto del jugador por *cat*.
- **unsigned Preguntas_Respondidas()**. Devuelve el número de preguntas respondidas por el jugador.
- **void Aumentar_Preguntas_Respondidas()**. Aumenta el número de preguntas respondidas del jugador en 1.
- **void Aumentar_Preguntas_Correctas()**. Aumenta el número de preguntas respondidas correctamente del jugador en 1.
- **double Rating()**. Devuelve el rating de respuesta del jugador.
- **void Mejores_Puntuaciones(vector<int>puntos)**. Devuelve en *puntos* las mejores puntuaciones del jugador en cada concurso.
- **void Actualizar_Puntuaciones(int Concurso, int Puntos)**. Actualiza la puntuación del concurso *Concurso* con los puntos *Puntos* si fuese necesario.
- **int Puntuacion()**. Devuelve la puntuación del jugador.
- **void SumaPuntos(int Puntos)**. Suma los puntos *Puntos* a la puntuación del jugador.
- **void RestaPuntos(int Puntos)**. Resta los puntos *Puntos* a la puntuación del jugador.

A continuación se muestra el diagrama de dependencias del módulo:

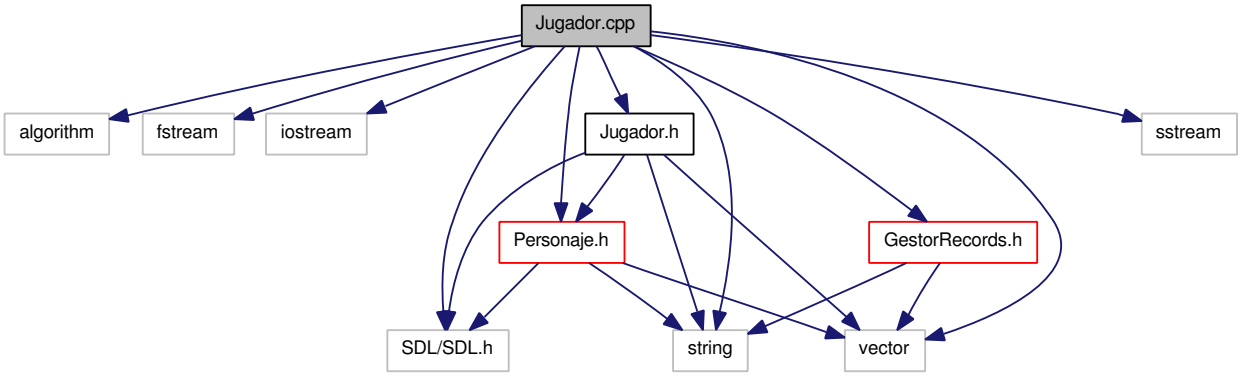

Figura 5.45: Diagrama de dependencia de Jugador

5.11. Personaje

Módulo encargado para gestionar los distintos jugadores.

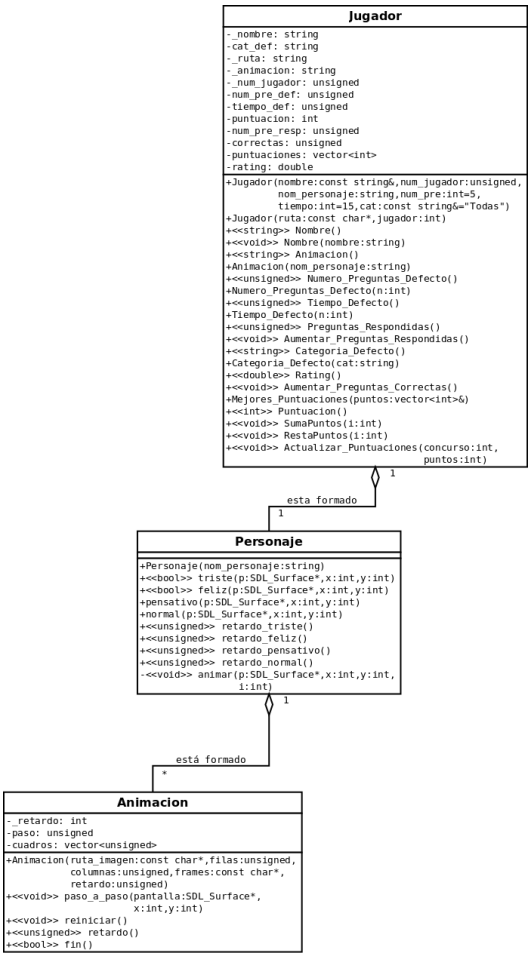

Figura 5.46: Diagrama de clases de la Clase Personaje

- **Personaje(string NomPersonaje)**. El constructor de la clase Personaje. Crea el personaje referenciado por *NomPersonaje*.
- **Personaje(const Personaje& j)**. Constructor de copia.
- **Personaje& operator =(const Personaje& j)**. Operador de asignación.
- **bool triste(Pantalla, int x, int y)**. Dibuja la animación triste del personaje en la posición x e y. Devuelve true si ha acabado la animación y false en caso contrario.
- **bool pensativo(Pantalla, int x, int y)**. Dibuja la animación pensativa del personaje en la posición x e y. Devuelve true si ha acabado la animación y false en caso contrario.
- **bool feliz(Pantalla, int x, int y)**. Dibuja la animación feliz del personaje en la posición x e y. Devuelve true si ha acabado la animación y false en caso contrario.
- **bool normal(Pantalla, int x, int y)**. Dibuja la animación normal del personaje en la posición x e y. Devuelve true si ha acabado la animación y false en caso contrario.
- **unsigned retardo_triste()**. Devuelve el retardo de la animación triste.
- **unsigned retardo_pensativo()**. Devuelve el retardo de la animación pensativo.
- **unsigned retardo_feliz()**. Devuelve el retardo de la animación feliz.
- **unsigned retardo_normal()**. Devuelve el retardo de la animación normal.

A continuación se muestra el diagrama de dependencias del módulo:

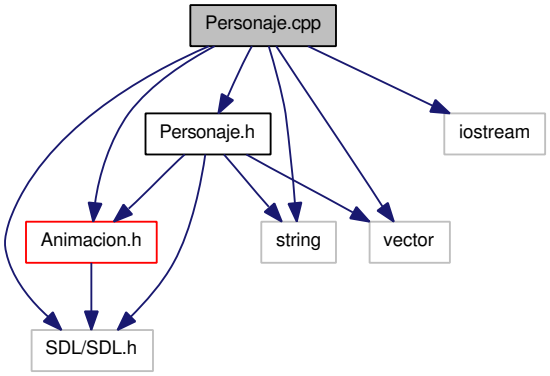

Figura 5.47: Diagrama de dependencia de Personaje

5.12. Animacion

Módulo encargado de las animaciones del videojuego.

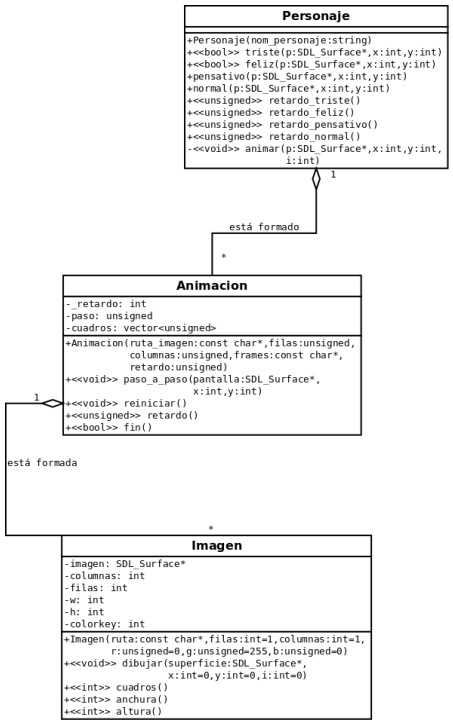

Figura 5.48: Diagrama de clases de la Clase Animacion

- **Animacion(string Ruta, unsigned Filas, unsigned Columnas, string Frames, unsigned Retardo)**. El constructor de la clase Animación. Crea una animación con la cuadricula referenciada por la ruta *Ruta*, que tiene *Filas* filas y *Columnas* columnas, en el orden dispuesto en *Frames* Ej:(̀0,1,4,7,8́) con retardo *Retardo*.
- **Animacion(const Animacion& j)**. Constructor de copia.
- **Animacion& operator =(const Animacion& j)**. Operador de asignación.
- **void paso_a_paso(Pantalla, int x, int y)**. Dibuja el siguiente paso de la animación en la posición *x* e *y*.
- **unsigned anchura()**. Devuelve la anchura de uno de los cuadros de la animación.
- **unsigned altura()**. Devuelve la altura de uno de los cuadros de la animación.
- **void reiniciar()**. Reinicia la animación.
- **unsigned retardo(void)**. Devuelve el retardo de la animación.
- **bool fin()**. Devuelve true si la animación a llegado a su último cuadro o false en caso contrario.

A continuación se muestra el diagrama de dependencias del módulo:

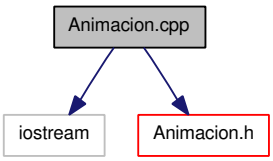

Figura 5.49: Diagrama de dependencia de Animacion

5.13. Imagen

Módulo encargado de manejar las imagenes.

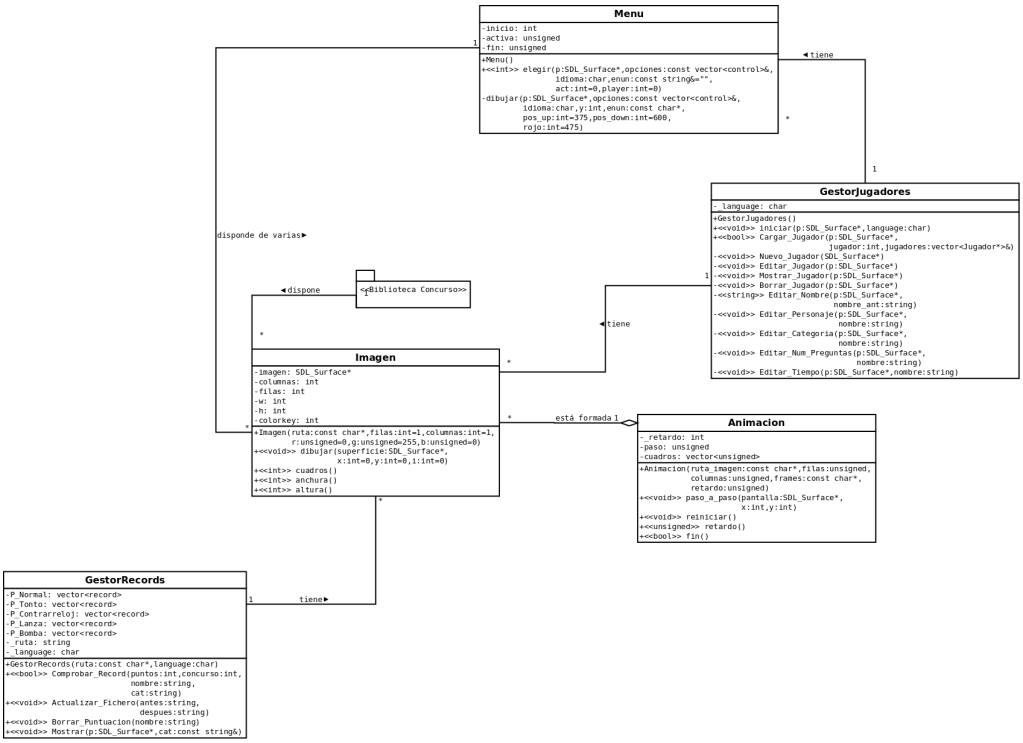

Figura 5.50: Diagrama de clases de la Clase Imagen

- **Imagen(string Ruta, int Filas, int Columnas, unsigned r, unsigned g, unsigned b)**. El constructor de la clase Imagen. Crea una cuadricula de imagenes referenciada por la ruta *Ruta*, que tiene *Filas* filas y *Columnas* columnas, y con color de transparencia representado por *r*,*g*,*b*.
- **Imagen(const Imagen& i)**. Constructor de copia.
- **Imagen& operator =(const Imagen& i)**. Operador de asignación.
- **void dibujar(Pantalla, int x, int y, int i)**. Dibuja el cuadro *i* en la posición *x* e *y*.
- **unsigned anchura()**. Devuelve la anchura de uno de los cuadros de la animación.
- **unsigned altura()**. Devuelve la altura de uno de los cuadros de la animación.
- **int cuadros(void)**. Devuelve el número de cuadros de la imagen.

A continuación se muestra el diagrama de dependencias del módulo:

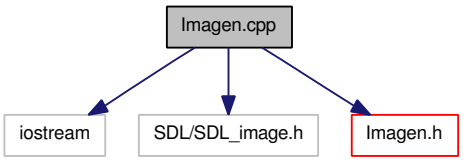

Figura 5.51: Diagrama de dependencia de Imagen

5.14. Sonido

Módulo encargado de los efectos sonoros del videojuego.

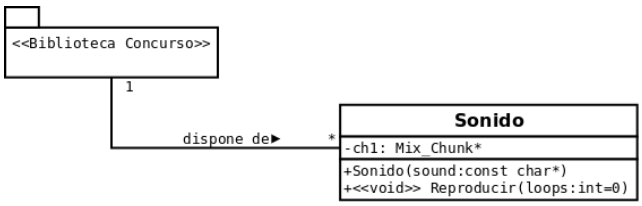

Figura 5.52: Diagrama de clases de la Clase Sonido

- **Sonido(string Sound)**. El constructor de la clase Sonido. *Sound* es la ruta del archivo de audio a reproducir.
- **void Reproducir(int loops)**. Reproduce el sonido. El parámetro *loops* indica las veces que se reproduce. -1 es infinito.

A continuación se muestra el diagrama de dependencias del módulo:

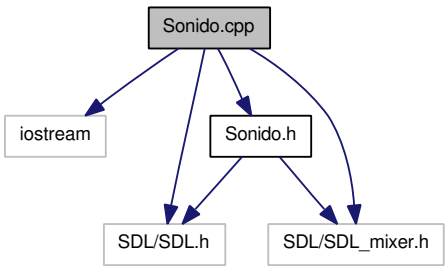

Figura 5.53: Diagrama de dependencia de Sonido

5.15. Musica

Módulo encargado de la música del videojuego.

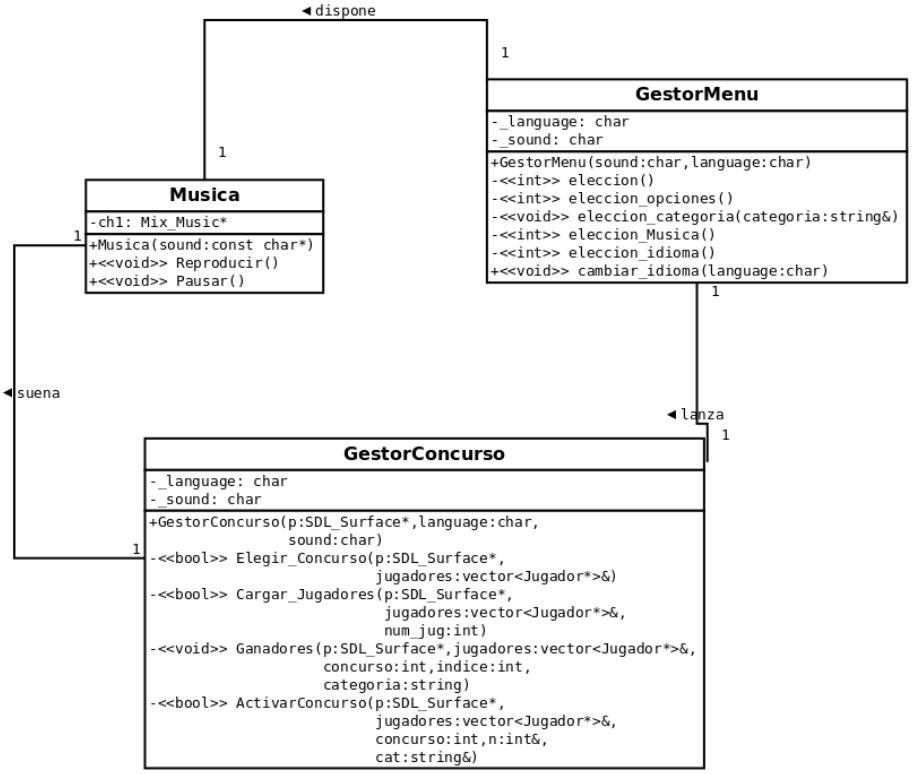

Figura 5.54: Diagrama de clases de la Clase Musica

- **Musica(string Sound)**. El constructor de la clase Musica. *Sound* es la ruta del archivo de audio a reproducir.
- **void Reproducir()**. Reproduce la música.
- **void Pausar()**. Pausa la música.

A continuación se muestra el diagrama de dependencias del módulo:

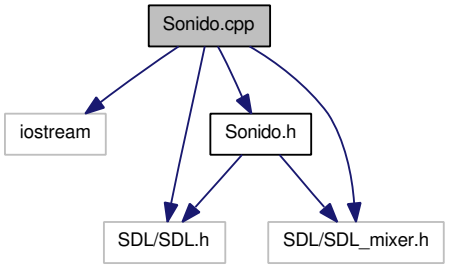

Figura 5.55: Diagrama de dependencia de Musica

Tiene el mismo diagrama que Sonido porque está en el mismo fichero.

5.16. Fuente

Módulo encargado de escribir por pantalla.

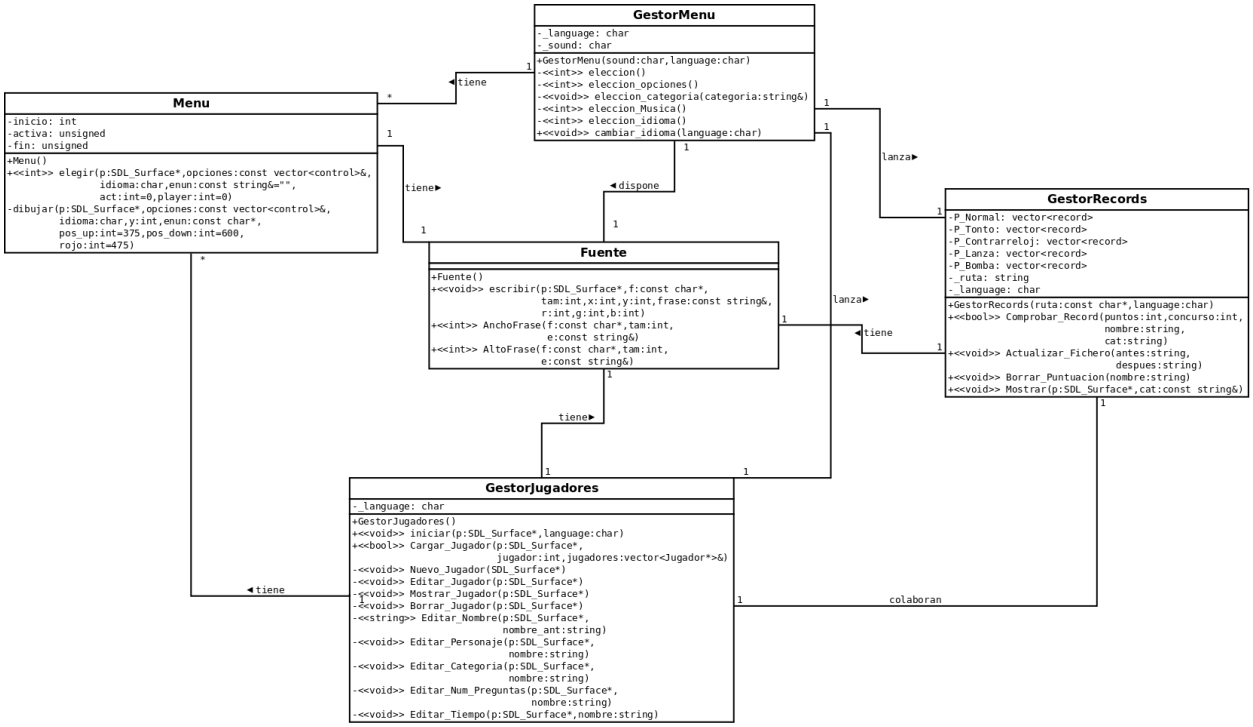

Figura 5.56: Diagrama de clases de la Clase Fuente

- **Fuente**(). El constructor de la clase Fuente.
- **void escribir(Pantalla, string f, int tam, int x, int y, string Frase, int r, int g, int b)**. Escribe la frase *Frase* de tamaño *tam* con la tipografía *f* en la posición *x* e *y* con el color representado por *r*,*g* textitb.
- **int AnchoFrase(string f, int tam, string e)**. Devuelve el ancho de la frase *e*, con tamaño *tam* y tipografía *f* en pixeles.
- **int AltoFrase(string char* f, int tam, string e)**. Devuelve el alto de la frase *e*, con tamaño *tam* y tipografía *f* en pixeles.

A continuación se muestra el diagrama de dependencias del módulo:

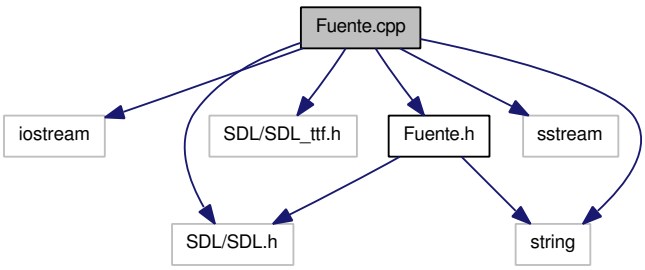

Figura 5.57: Diagrama de dependencia de Fuente

5.17. Video

Módulo encargado de inicializar el subsistema de Video.

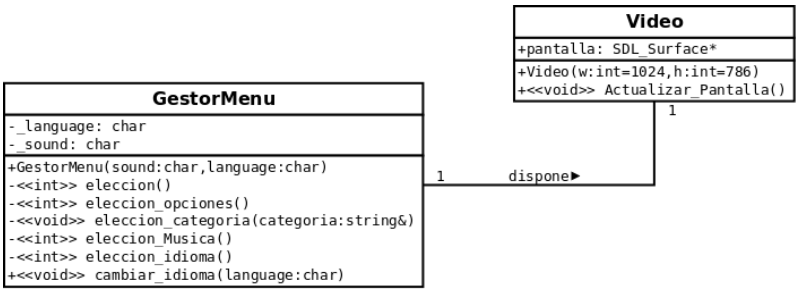

Figura 5.58: Diagrama de clases de la Clase Video

- **Video(int w, int h)**. El constructor de la clase Video. Crea una superficie de tamaño *w* por *h*
- **void Actualizar_Pantalla()**. Actualiza la superficie de video.

A continuación se muestra el diagrama de dependencias del módulo:

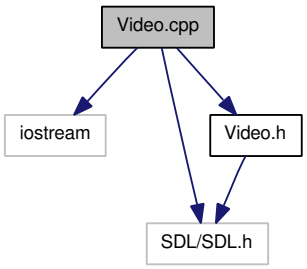

Figura 5.59: Diagrama de dependencia de Video

5.18. Inicio

Es la encargada de poner en marcha la biblioteca SDL.

Figura 5.60: Diagrama de clases de la Clase Inicio

- **Inicio()**. El constructor de la clase Inicio.

A continuación se muestra el diagrama de dependencias del módulo:

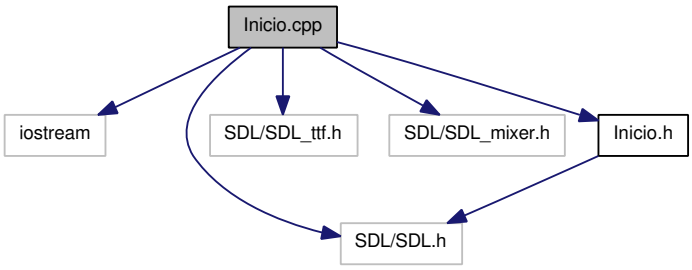

Figura 5.61: Diagrama de dependencia de Inicio

5.19. Características de los usuarios finales

El videojuego se ha diseñado pensando en que los usuarios finales sean los que determinen la dificultad y los campos de conocimientos que se necesiten para poder jugar al mismo. El videojuego está diseñado para cualquier tipo de jugador con ganas de divertirse y aprender, únicamente deberá familiarizarse con los buzzers.Va dirigido a cualquier edad, ya que se trata de un juego de fácil de usar y permite la creación de las preguntas deseadas.

Capítulo 6

Implementación

Una vez que hemos finalizado las fases de análisis y diseño, sólo nos queda codificar todas las instrucciones usando un lenguaje de programación y usando una herramienta gráfica. En nuestro caso hemos elegido el lenguaje de programación C++ y la librería gráfica libSDL.

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje multiparadigma. Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales. C++ permite trabajar tanto a alto como a bajo nivel siendo muy óptimo.

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre “C con clases”. En C++, la expresión “C++” significa “incremento de C” y se refiere a que C++ es una extensión de C.

El motivo de la elección de C++, es el grado de familiarización con este tipo de lenguaje es muy alto. En la Universidad de Cádiz es el lenguaje por el cual se rigen la mayoría de las asignaturas. Es un lenguaje muy eficiente y que nos ayuda con multitud de herramientas que nos serán útiles a lo largo de la fase de implementación.

Una de esas herramienta es la libSDL, Simple DirectMedia Layer (SDL), que es un conjunto de bibliotecas desarrolladas con el lenguaje C que proporcionan funciones básicas para realizar operaciones de dibujado 2D, gestión de efectos de sonido y música, y carga y gestión de imágenes [4].La biblioteca se distribuye bajo la licencia LGPL, que es la que ha provocado el gran avance y evolución de las SDL. Se han desarrollado una serie de bibliotecas adicionales que necesitamos ya que extienden las funcionalidades de la librería:

- SDL Mixer: Extiende las capacidades de SDL para la gestión y uso de sonido y música en aplicaciones y juegos. Soporta formatos de sonido como Wave, MP3 y OGG, y formatos de música como MOD, S3M, IT, y XM.
- SDL Image: Extiende notablemente las capacidades para trabajar con diferentes formatos de imagen. Los formatos soportados son los siguientes: BMP, JPEG, TIFF, PNG, PNM, PCX, XPM, LBM, GIF, y TGA,

- SDL TTF: Permite usar fuentes TrueType en aplicaciones SDL.

Si tuviera que dar un motivo por el cual elegir esta herramienta sería que dicha herramienta es familiar para mí, ya que al haber cursado la asignatura de Diseño de Videojuegos, cuyo profesor es el tutor de este proyecto, usamos la SDL para desarrollar el proyecto de la asignatura. Por tanto, usando los documentos y tutoriales que facilita la asignatura es bastante fácil aprender a usar dicha librería. Además, es una librería simple y que da bastante buenos resultados, siempre que nos dediquemos a usarla en un juego en dos dimensiones, ya que si el proyecto fuese en tres dimensiones tendríamos que usar OpenGL[1] y una librería como Ogre[2].

6.1. Desarrollo de Prerres

Para la compilación usamos el compilador de C++ llamado g++, que viene incluída en la distribución de cualquier sistema GNU/Linux.

El proyecto se divide en núcleo del juego y las pruebas que lo componen. El núcleo del juego lo componen la interfaz gráfica de los menús principales, la gestión de jugadores y la gestión de records.

6.1.1. Núcleo del Juego

En primer lugar hablaremos de la interfaz gráfica. La interfaz gráfica se componen de imágenes desarrolladas en el programa GIMP, recogidas de bancos de imagenes y botones creados por mí. Todo ello, mediante la librería libSDL, es lo que compone la interfaz gráfica.

También hablaremos del desarrollo de la biblioteca de preguntas en formato GIFT. El desarrollo de la la biblioteca se basa en la lectura de un fichero que se genera mediante el script en perl Gift-0.6. Al leer ese fichero, se va seleccionando la información de las diferentes preguntas y se van almacenando en memoria mediante una clase. Por tanto, las operaciones que necesitará el módulo Lote será leer el fichero especificado y guardar su contenido en forma de clases.
Una de las partes esenciales del juego es la gestión de jugadores. Esto nos permite guardar datos sobre los jugadores en memoria secundaria para que a cada ejecución del software no se empiece siempre desde el principio. En cada jugador se va a poder guardar su nombre, personaje, categoría de preguntas, número de preguntas, tiempo de contrarreloj, número de preguntas respondidas por el jugador, rating de acierto en las respuestas y la mejor puntuación de ese jugador en cada concurso.
También se va a llevar un control de las mejores puntuaciones de cada concurso por la categoría de las preguntas. Se guardarán las 4 mejores puntuaciones de cada concurso para cada categoría del mismo. Esto será flexible, ya que si se borra un jugador o se borra una categoría, tambien se borrarán los datos que se guarde en la gestión de records. Del mismo modo si se crean nuevas categorías el software está capacitado para incluirlas y guardar las puntuaciones de los concursos.

[1] OpenGL (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos.

[2] OGRE 3D (acrónimo del inglés Object-Oriented Graphics Rendering Engine) es un motor de renderizado 3D orientado a escenas, escrito en el lenguaje de programación C++.

Sus librerías evitan la dificultad de la utilización de capas inferiores de librerías gráficas como OpenGL y Direct3D, y además, proveen una interfaz basada en objetos del mundo y otras clases de alto nivel.

6.1.2. Concursos de Prerres

En segundo lugar hablaremos de los distintos concursos que componen el proyecto. Cada concurso se corresponde con una clase, excepto el concurso **Gran Concurso**, aunque todos están controlados por la clase GestorConcurso, que es la encargada tanto de iniciar como de terminar los concursos. Cada concurso posee su propio game loop, es decir, el bucle principal del juego en el que se controlan todos los eventos del concurso, y además tendrán sus funciones y procedimientos privados.

La mayoría de los concursos tienen una inicialización de datos muy parecida. Lo que realmente cambia de uno a otro es la forma de interactuar con el usuario y con el tiempo.
Inicialmente cuando creamos un concurso de cualquier tipo, el sistema:

1. Crea la biblioteca de preguntas, o lo que es lo mismo, un objeto de la clase Lote.
2. Carga las imagenes necesarias en el concurso: fondos, cuadros, imagenes, ...
3. Crea un vector de clase Jugador de tamaño igual al número de jugadores elegido por el usuario.
4. Guarda la categoría por defecto del jugador 1 en una variable de tipo string.
5. Dependiendo de si es un concurso simple (**Normal**, **Pasa la Bomba**, **Tonto el Último** o **Lanza la pregunta**) o si es el concurso **Contrarreloj**, se guarda el número de preguntas o el tiempo de contrarreloj por defecto del jugador 1 en una variable de tipo entero.

A partir de aquí, para cada tipo de concurso actua de forma diferente. Aunque lo que realmente diferencia los tipos de concursos es la interacción con los jugadores y la forma en la que se tratan las preguntas.
A continuación, se explica como se han implementado los diferentes concursos.

Normal

Para el desarrollo de este concurso se ha creado un bucle en el que:

1. Se calcula el jugador que le toca el turno.
2. Se actualiza el estado de los jugadores.
3. Se saca una pregunta del Lote con las características que correspondan.
4. Se escribe en pantalla.
5. Se ponen en marcha el cronometro del jugador.
6. Se entra en el game loop a esperar que el usuario responda a la pregunta.
7. Se reinicia el cronometro del jugador.
8. Se contabiliza la puntuación del jugador y se destruye la pregunta.

Este bucle se ejecuta tantas veces como turnos alla, dependiendo del número de jugadores y del número de preguntas.
Una vez se acaba el bucle, se pasa el control a la clase GestorConcurso, que será la encargada de dar los ganadores y realizar las tareas de actualización de records y jugadores.
El game loop del concurso **Normal** sigue el siguiente procedimiento:

1. Se inicializan variables.

2. Se abren los buzzers.
3. Se inicializan las animaciones de los personajes.
4. Se entra en un bucle de control de eventos.
    a) Se actualiza el cronometro.
    b) Se comprueba si el cronometro ha llegado al final.
        1) Si ha llegado al final.
            a' Se comprueba si la respuesta seleccionada es la correcta.
            b' Se guarda la información y se sale del bucle.
    c) Se actualizan las animaciones de los personajes.
    d) Se comprueban los eventos de joystick y de teclado.
        1) Si se ha pulsado un botón o tecla.
            a' Se comprueba que es el mando del jugador que tiene el turno o teclas activas para el jugador.
            b' Se selecciona la respuesta y se muestra por pantalla.
            c' Se comprueba si es correcta o no la respuesta y se muestra en pantalla.
            d' Se sale del bucle de control de eventos.
5. Una vez fuera del bucle, se cierra los buzzers.
6. Se muestran las animaciones correspondientes.
7. Se liberan los recursos.

Este básicamente es el game loop de este concurso.

Contrarreloj

Para el desarrollo de este concurso se ha creado un bucle en el que:

1. Se calcula el jugador que le toca el turno.
2. Se actualiza el estado de los jugadores.
3. Se saca una pregunta del Lote con las características que correspondan.
4. Se ponen en marcha el cronometro del jugador.
5. Se dibuja en pantalla.
6. Se entra en el game loop a esperar que el usuario responda a la pregunta.
7. Se guarda el tiempo de cronometro del jugador.
8. Se contabiliza la puntuación del usuario y se destruye la pregunta.

Este bucle se ejecuta hasta que todos los jugadores hayan terminado su tiempo, prefijado en las opciones de concurso.
Una vez se acaba el bucle, se pasa el control a la clase GestorConcurso, que será la encargada de dar los ganadores y realizar las tareas de actualización de records y jugadores.
El game loop del concurso **Contrarreloj** es idéntico al gameloop del concurso **Normal**.

Pasa la Bomba

Para el desarrollo de este concurso se ha creado un bucle en el que:

1. Se calcula el tiempo de la bomba.
2. Se actualiza el estado de los jugadores.
3. Se ponen en marcha el tiempo de la bomba.
4. Se saca una pregunta del Lote con las características que correspondan.
5. Se escribe en pantalla.
6. Se entra en el game loop a esperar que el usuario responda a la pregunta.
7. Se contabiliza la puntuación del jugador y estadísticas.
8. Si explota la bomba.
    a) Se dibuja la explosión.
    b) Se calcula el jugador que le toca el turno.
9. Si no explota la bomba.
    a) Si el jugador acierta la pregunta.
        1) Se pasa el turno al siguiente jugador.
    b) En caso contrario.
        1) Se vuelve al paso 4.
10. Se destruye la pregunta.

Este bucle se ejecuta tantas veces como bombas alla, dependiendo del número de preguntas.
Una vez se acaba el bucle, se pasa el control a la clase GestorConcurso, que será la encargada de dar los ganadores y realizar las tareas de actualización de records y jugadores.
El game loop del concurso **Pasa la Bomba** sigue el siguiente procedimiento:

1. Se inicializan variables.
2. Se abren los buzzers.
3. Se inicializan las animaciones de los personajes.
4. Se entra en un bucle de control de eventos.
    a) Se actualiza el cronometro.
    b) Se comprueba si el cronometro ha llegado al final.
        1) Si ha llegado al final.
            a' Es que ha explotado la bomba.
            b' Se sale del gameloop.
    c) Se actualizan las animaciones de los personajes.
    d) Se comprueban los eventos de joystick y de teclado.
        1) Si se ha pulsado un botón o tecla.

a' Se comprueba que es el mando del jugador que tiene el turno o teclas activas para el jugador.

b' Se selecciona la respuesta y se muestra por pantalla.

c' Se comprueba si es correcta o no la respuesta y se muestra en pantalla.

d' Se sale del bucle de control de eventos.

5. Una vez fuera del bucle, se cierra los buzzers.
6. Se muestran las animaciones correspondientes.
7. Se liberan los recursos.

Este básicamente es el game loop de este concurso.

Tonto el Último

Para el desarrollo de este concurso se ha creado un bucle en el que:

1. Se actualiza el estado de los jugadores.
2. Se saca una pregunta del Lote.
3. Se ponen en marcha los cronómetros de los jugadores.
4. Se dibuja en pantalla.
5. Se entra en el game loop, y cada jugador elige una respuesta.
6. Se reinician los cronómetros.
7. Se actualizan las puntuaciones y se destruye la pregunta.

Este bucle se ejecuta tantas veces como número de preguntas se haya seleccionado en las opciones del concurso.
Una vez se acaba el bucle, se pasa el control a la clase GestorConcurso, que será la encargada de dar los ganadores y realizar las tareas de actualización de records y jugadores.
El game loop del concurso **Tonto el Último** sigue el siguiente procedimiento:

1. Se inicializan variables.
2. Se abren los buzzers.
3. Se inicializan las animaciones de los personajes.
4. Se entra en un bucle de control de eventos.

   a) Se actualiza el cronometro.

   b) Se comprueba si el cronometro ha llegado al final.

      1) Si ha llegado al final.

         a' Se comprueba el orden de las respuestas.

         b' Se comprueba cuales son correctas y cuales no.

         c' Se asignan los puntos que correspondan.

         d' Se guarda la información y se sale del bucle.

c) Se actualizan las animaciones de los personajes.

d) Se comprueban los eventos de joystick y teclado.

1) Si se ha pulsado un botón o tecla.

a' Se comprueba que es un mando activo o teclas activas.

b' Se cambia la respuesta por la escogida por el jugador.

c' Se actualiza el orden de contestación.

5. Una vez fuera del bucle, se cierra los buzzers.
6. Se muestran las animaciones correspondientes.
7. Se liberan los recursos.

Este básicamente es el game loop de este concurso.

Lanza la Pregunta

Para el desarrollo de este concurso se ha creado un bucle en el que:

1. Se calcula el jugador que le toca el turno.
2. Se actualiza el estado de los jugadores.
3. Se saca una pregunta del Lote con las características que correspondan.
4. Se dibuja en pantalla.
5. Se espera que el jugador lance una pregunta a otro jugador.
6. Se pone en marcha el cronómetro del jugador elegido.
7. Se entra en el game loop a esperar que el usuario responda a la pregunta.

a) Si el jugador falla la pregunta.

1) Pasa el turno al siguiente jugador.
2) Vuelve a 5.

8. Se reinician los cronómetros.
9. Se actualizan las puntuaciones.
10. Se reinicia el cronometro del jugador.
11. Se actualizan las puntuaciones y se destruye la pregunta.

Este bucle se ejecuta tantas veces como número de preguntas se haya seleccionado en las opciones del concurso.
Una vez se acaba el bucle, se pasa el control a la clase GestorConcurso, que será la encargada de dar los ganadores y realizar las tareas de actualización de records y jugadores.
El game loop del concurso **Lanza la Pregunta** es idéntico al gameloop del concurso **Normal**, con la salvedad comentada antes de que si falla la respuesta, el turno pasa al jugador siguiente con la misma pregunta.

6.1.3. Gran Concurso

Este tipo de concurso se compone de los demás por lo que está implementado en la clase GestorConcurso. Lo que hace unicamente es guardar que concursos debe ir ejecutando y en que orden, manteniendo los jugadores y su información hasta que se ejecuten todos los concursos elegidos.
Para ello tenemos un vector de id de concurso, que se les van pasando a la función encargada de lanzar los concursos. En cada ejecución de concurso nos quedamos con el vector jugadores que contiene los jugadore y su información para pasarselo al siguiente concurso. Así conseguimos que se mantengan los jugadores y su información de un concurso a otro.

6.1.4. Menús

El desarrollo de los menús del juego, se ha hecho de tal manera, que puede ser utilizado en todos los menús que se presentan a lo largo de la aplicación. Con esto se ahorra tiempo y se reutiliza código.
La clase Menu, recibe un vector con las opciones del menú e indicandole una serie de opciones como el idioma de las opciones, la superficie donde dibujarlo, si precisa de algún enunciado o la opción elegida por defecto, dibuja el menú y espera a qué el usuario escoga una de las opciones devolviendo su número de opción, que se determina por su posición en la lista de opciones.
Este diseño de menús está especialmente pensado para los mandos buzzers. Con los botones que se indican en pantalla se sube y se baja en las opciones y se escoge la opción que queda a la altura del botón rojo cuando este es pulsado.

6.1.5. Gestión de Jugadores

Para gestionar los jugadores lo que se ha creado es una serie de ficheros tanto de cada jugadores como de todos los jugadores. Los ficheros de cada jugador llevan el nombre del jugador seguido de la extensión .sav. Estos ficheros guardan información por líneas de el nombre, personaje, categoría de preguntas, número de preguntas del concurso por defecto, tiempo de contrarreloj por defecto, número de preguntas acertadas, número de preguntas respondidas, mejor puntuación concurso **Normal**, mejor puntuación concurso **Contrarreloj**, mejor puntuación concurso **Pasa la Bomba**, mejor puntuación concurso **Tonto el Último** y mejor puntuación concurso **Lanza la Pregunta**.
Luego se guarda otro fichero en el cual se tienen los nombres de los jugadores que existen registrados en el videojuego. Esto sirve de control a la hora de cargar los diferentes jugadores.
A la hora de crear, actualizar o borrar jugadores unicamente debemos crear, actualizar o borrar dichos ficheros, por lo que la gestión de jugadores se hace sencilla. Aparte, si borramos un jugador o cambiamos su nombre, tendremos que llamar a la clase de Gestión de Records para actualizar los datos que tuviese guardados de ese jugador.

6.1.6. Gestión de Records

Para la gestión de records se ha creado un fichero en el que se guarda por cada categoria las 4 mejores puntuaciones de cada concurso, este fichero debe estar siempre actualizado y debe ser sensible a los cambios de nombre de los jugadores, adición o eliminación de categorías y eliminación de jugadores. Este fichero se llama record.sav y se puede encontrar en la subcarpeta data.
Para ello, utilizamos la clase GestorRecords que se encarga de todo lo descrito en el parrafo anterior.

6.1.7. Motor del juego

Así denomino a todas las clases auxiliares que permiten controlar el sonido, el video, las animaciones y demás módulos que pueda tener un juego. En nuestro caso, disponemos de las siguientes clases:

- Inicio
- Animacion
- Cronometro
- Jugador
- Personaje
- Fuente
- Imagen
- Sonido
- Musica
- Video

Para más información sobre estas clases y su implementación, se recomida consultar la documentación generada por doxygen en la carpeta */prerres/doc/*.

Capítulo 7

Pruebas

En este apartado presentaremos como hemos procedido a realizar las pruebas a los distintos componentes del juego.

7.1. Pruebas realizadas

Según las recomendaciones de G. J. Myers [5]:

1. Cada caso de prueba debe especificar el resultado esperado: Es decir el juego funciona correctamente y devuelve una puntuación válida.
2. El programador debe evitar probar sus propios programas: Los usuarios elegidos para probar el proyecto siempre son personas ajenas al juego.
3. Se debe analizar con detalle el resultado de cada prueba: Se debe comprobar con detenimiento que todas las funciones de cada concurso funcionan correctamente y muestran el ganador de forma correcta.
4. Se deben incluir entradas válidas e inválidas: El usario externo al proyecto prueba todos los mandos y botones de los mismos y se comprueba que no producen efectos no deseados.
5. Se debe:
   - Probar si el software no hace lo que debe. El juego debe captar los eventos necesarios y cargar todos los escenarios que hacen falta.
   - Probar si el software hace lo que no debe. El juego se vuelve inestable en algún caso en el que debería funcionar correctamente.
6. Los casos de prueba deben documentarse: Mostraremos las pruebas realizadas en cada iteración más adelante.
7. Hay que asumir que hay errores: En el momento en el que programamos códigos largos suele encontrarse muchos fallos.
8. Donde hay un defecto hay otros: En caso de que encontremos un fallo, ese fallo puede provocar otros más o menos graves que éste.
9. Las pruebas son una tarea creativa: No nos debemos quedar en las pruebas ideales, debemos probar el programa con todo tipo de casos y situaciones.

7.1.1. Incremento 1: Requisitos Básicos del sistema

¿El sistema carga correctamente los fondos y aparece la pantalla principal?

El sistema desde el primer momento carga los fondos, los botones después de unos ajustes son colocados en sus posiciones correspondientes.

¿El sistema capta correctamente la pulsaciones de los distintos botones del mando y de las teclas?

El programa capta correctamente la pulsación de los botones de los mandos habilitados y las teclas para los menús.

¿Los menús se muestran de forma correcta?

Los menús se muestran correctamente en pantalla y es posible elegir las opciones de forma correcta.

7.1.2. Incremento 2: Organización de las preguntas

¿Se puede crear y destruir objetos de la clase Pregunta, MultiOpcion, Corta, Numerica, TrueFalse, Relacion y Lote?

Se prueba en cada una de las clases los constructores y destructores de las mismas, teniendo el funcionamiento deseado en todas. La única que daba problemas era la clase Lote ya que en un principio el destructor no liberaba bien los recursos, pero era un fallo de implementación, ya que no liberaba los punteros guardados en los vectores que forman parte de la clase.

¿Las funciones observadoras devuelven lo deseado?

Se prueban las funciones observadoras y hay que arreglar ciertos detalles, como formato de devolución para las clases MultiOpcion, Corta, Numerica, TrueFalse y Relación. Las observadores de la clase Lote también tiene un problema. La primera versión de las funciones que devolvían las preguntas tenía el mismo problema que el destructor, no liberaba la memoría de la pregunta que acababa de devolver. Esto hacía que la memoría principal no parara de subir y se abortaba el programa. Hablandolo con compañeros y buscando por internet, pudé arreglar este problema y todo funcionaba de forma correcta.

Una de las complicaciones que aparece más adelante con estas clases es que poseían demasiada información. Al crearse muchos objetos, hacía que el tamaño en memoria fuera desorbitado. Hubo que implementar las relaciones entre clases en las propias funciones en vez de ser atributos de la clase. Con esto el espacio en memoria se consiguió reducir a la mitad.

¿Las funciones de las clases Pregunta, MultiOpcion, Corta, Numerica, TrueFalse, Relacion y Lote funcionan de forma correcta?

Se prueban las funciones de las distintas clases. Hay que modificar la función de escribir de las clases MultiOpcion, Corta, Numerica, TrueFalse y Relacion para que todas salga como es debido. Los constructores de copia de estás mismas clases también deben ser revisados al pasar por alto uno de los parámetros a copiar.

¿Se guardan correctamente las preguntas en la clase Lote con el algoritmo de transformación de preguntas?

La parte que más pruebas ha necesitado ha sido la del algoritmo que guarda las preguntas. Se conseguía que se guardarán la mayoría pero había algunas que no se guardaban de forma correcta. Después de muchas modificaciones de código se consigue que el sistema funcione perfectamente. Para ello, intentamos insertar por tipos de preguntas. Primero se insertaban las multiopción, luego las multirespuesta, etc. Así se descubría en que tipo de pregunta estaba el fallo. Una vez solucionado se volvía a probar con las preguntas mezcladas por si había efectos colaterales. Evidentemente si que los había y se solucionaron despues de hacer una minuciosa traza sobre la ejecución de la función: fichero de preguntas, lo que iba leyendo y guardando, etc.

7.1.3. Incremento 3: Diseño e Implementación de clases básicas

¿Se inicia correctamente la SDL?

Se prueba si la SDL se inicia de forma correcta y si, ya que sino daría un error al iniciarse.

¿Se inicia correctamente el subsistema de video y el de sonido?

El subsistema de video también se inicia de forma correcta ya que podemos ver la ventana con la pantalla principal. El de sonido igual, ya que podemos escuchar la música del menú principal.

¿Se carga una imagen de forma correcta?

Las imagenes se cargan de forma correcta y se liberan también de forma correcta.

¿Se escribe todo tipo de frases de forma correcta por pantalla?

Se puede escribir frases en la pantalla pero si es muy larga hay problema de presentación. Para ello se modifica el código para que la presentación para frases más larga de una línea mejore. Una vez realizados estos cambios se puede escribir cualquier frase por pantalla sin que se salga ni quede mal presentado.

7.1.4. Incremento 4: Diseño e Implementación de la clase GestorMenu

¿Se activa o desactiva de forma correcta el sonido?

La primera opción que se prueba es la del sonido. Simplemente lo que se hace es pausar la música hasta que se vuelva a reactiva por el usuario. Hacemos la prueba y podemos desactivarlo como activarlo de forma correcta. Además si pulsamos dos veces seguidas en la misma opción no tiene efectos indeseados.

¿Se cambia el idioma de forma correcta?

Hay dos idiomas disponibles, español e inglés. Al cambiar el idioma se guarda por defecto y la próxima vez que se ejecute el juego se muestra el último idioma escogido. Probamos que se puede cambiar el idioma en la misma ejecución tantas veces como se desee, que si pulsamos dos veces seguidas en la misma opción no tiene efectos indeseados y reiniciamos el juego con las dos opciones de idiomas y se

cargan correctamente.

¿Se cargan las nuevas preguntas de forma correcta?

Al iniciar el programa se genera el fichero Preguntas.gift y cada vez que se pulse la opción de actualizar preguntas se vuelve a crear el archivo. Para probarlas entonces, ejecutamos el juego con el archivo de preguntas vacío y comprobamos que el fichero Preguntas.gift está vacío. Introducimos preguntas en el archivo de preguntas mientras el juego está en ejecución y elegimos la opción de actualizar las preguntas. Comprobamos que el fichero Preguntas.gift, dispone de las preguntas que hemos introducido en el otro fichero en formato GIFT.

¿Funcionan los menús despues de la reestructuración de los mismos? ¿Responden bien los mandos a las pulsaciones de botones y a las teclas?

Debido a la elección de los buzzers como control del juego, se reestructuran los menús. Estos ahora sólo muestran 3 opciones y como seleccionada la que está en el centro. El usuario debe situar en el centro, mediante los diferentes botones, la opción que desee. Esto creaba efectos indeseados ya que la primera y la último opción no se mostraba de forma correcta por problemas con los indices de los vectores. Una vez arreglado este problema, las opciones del menú se presentan de forma correcta en todas sus combinaciones.

En los menús también había un problema de rendimiento. Cuando no se estaban escogiendo opciones la CPU estaba trabajando al 100 %. Esto tenía su origen en la forma de recoger los eventos. Se consiguió solucionar durmiendo el proceso hasta que el jugador no pulsara un botón. Esto hizo que el gasto de CPU disminuyera de 100 % a 3 %.

Cuando todavía no se ha accedido a ningún concurso sólo el primer jugador puede moverse entre los menús para que no sea un descontrol al todos poder moverse por los menús. Probamos que sólo el primer jugador se puede mover por los menús y que ningún otro jugador puede mover o seleccionar ninguna opción. Se comprueba que funciona de forma correcta.

El mismo efecto produce para las teclas, ya que utiliza el mismo código.

7.1.5. Incremento 5: Planificación de concursos e implementación de clases

¿Se construyen y destruyen correctamente objetos de las clases Jugador, Cronometro, GestorConcurso, Concurso, Simple y Contrarreloj?

Todo funciona de forma correcta. Se pueden crear y destruir objetos de todas las clases sin problemas.

¿Se ejecutan de forma correcta, devolviendo los datos correctamente, las funciones observadoras de las clases Jugador, Cronometro?

Las de la clase jugador sin problema. La clase cronometro tenía problemas a la hora de mostrar los segundos cuando el cronometro estaba en pausa. Era un problema de diseño que se solucionó de forma rápida, rediseñando el sistema de pausa.

¿Se actualizan los datos del jugador de forma correcta en la clase Jugador, mediante las funciones correspondientes?

Si, sin problemas.

¿Las funciones de control de la Clase Cronometro funcionan adecuadamente?

Como se comentó antes, la función de pausa, daba problemas porque no guardaba de forma correcta el tiempo que llevaba el cronometro. Además esto generaba efectos colaterales a las demás funciones de control, como era parar, o iniciar el cronometro. La solución se ha comentado antes.
¿Se selecciona de forma correcta la categoría por defecto de las preguntas?

Si.

¿Se selecciona correctamente el número de preguntas por defecto para los concursos simples?

Si.

¿Se selecciona correctamente el tiempo de contrarreloj por defecto para el concurso ***Contrarreloj****?*

Si.

¿Se cargan de forma correcta los jugadores para comenzar el concurso?

Si.

¿Se puede navegar de forma correcta por los menús del área de concursos?

No, hubo problemas con la música, ya que se paraba o se volvía a iniciar haciendo efecto de sonidos indeseados. Una vez estudiado donde iniciar y parar la música, no tenemos problemas a la hora de navegar por los menús.
¿Se inician de forma correcta los concursos y se pueden cambiar sus opciones?

Si.

¿Se muestran los ganadores de forma correcta mediante la clase GestorConcurso?

Si.

7.1.6. Incremento 6: Diseño e Implementación Concurso Normal

¿Se carga el juego de forma correcta?

Si.

¿Funciona correctamente para 1, 2, 3 o 4 jugadores?

No hay diferencias de funcionamiento entre distinto número de jugadores.

¿Se contabilizan bien las puntuaciones al responder una pregunta?

Si.

¿Se contabilizan bien las puntuaciones cuando se acaba el tiempo?

No. No se tiene en cuenta actualizar puntuación cuando se acaba el tiempo. Se rediseña la función y se prueba, obteniendo el resultado esperado.

¿Se señala acertadamente la respuesta correcta a la pregunta formulada?

Si.

7.1.7. Incremento 7: Diseño e Implementación Concurso Contrarreloj

¿Se carga el juego de forma correcta?

Si.

¿Funciona correctamente para 1, 2, 3 o 4 jugadores?

Si.

¿Se contabilizan bien las puntuaciones al responder una pregunta?

Si.

¿Se contabilizan bien las puntuaciones cuando se acaba el tiempo?

Si.

¿Se guarda de forma correcta el tiempo restante de los jugadores?

No. Entre cambios de jugador, había un retardo que hacía que se perdieran unos cuantos segundos. Para no tener que realizar una función por concurso para elegir pregunta, considero realizar una función para reajustar el cronometro desde fuera de esa función. Entonces sumo el retardo que tenga en cambiar de jugador al cronometro del jugador anterior. Esto soluciona el problema de una forma fácil.

¿Se señala acertadamente la respuesta correcta a la pregunta formulada?

Si.

¿Se termina el concurso cuando todos los jugadores acaban su tiempo?

No. Se terminaba el concurso una vez que un jugador llegaba al final de su tiempo. Era un problema de implementación que se arregla facilmente.

7.1.8. Incremento 8: Inclusión de animaciones

¿Se crean y destruyen correctamente los objetos de la clase Personaje?

Si. no hay problemas.

¿Funciona correctamente las funciones de la clase Personaje?

Si, sólo hay que rediseñar una de las funciones para que se pueda elegir entre animación que se reinicia y que no, ya que sino siempre se reinicia.

¿Una vez integrado con la clase Jugador, se muestran correctamente las animaciones desde la clase Jugador?

Si, de igual forma que si estuviera fuera de la clase.

7.1.9. Incremento 9: Diseño e Implentación de Concurso Tonto el Último

¿Se carga el juego de forma correcta?

Si.

¿Funciona correctamente para 2, 3 o 4 jugadores?

Inicialmente no se seleccionaban de forma correcta las respuestas, pero se debía a un fallo en los indices de los vectores. Una vez que si se escogían no funcionaba correctamente el sistema de orden de contestación que se expone más detalladamente luego.

¿Se contabilizan bien las puntuaciones al responder una pregunta?

No. No se respeta el orden de contestación debido a la inicialización del vector que se utiliza para tla fin, una vez rediseñado, el sistema funciona de forma correcta y se asignan correctamente las puntuaciones a cada jugador, según su respuesta escogida.

¿Puede cualquier usuario cambiar de respuesta durante el tiempo?

Si.

¿Se señala acertadamente la respuesta correcta a la pregunta formulada?

Si. En caso de que ningún jugador acierte la pregunta, también se señala la respuesta de forma correcta.

7.1.10. Incremento 10: Diseño e Implementación de Concurso Lanza Pregunta

¿Se carga el juego de forma correcta?

Si.

¿Funciona correctamente para 3 o 4 jugadores?

En principio no se asignaba de forma correcta los equipos y también cualquier jugador podía enviar la pregunta al equipo que quisiese. Esto se soluciona en primer lugar, arreglando los indices del vector que lleva el orden de los jugadores y en segundo lugar, bloqueando los mandos de los demás concursantes para que sólo el que tiene el turno pueda mandar la pregunta.

¿Se contabilizan bien las puntuaciones al responder una pregunta?

Si.

¿Se contabilizan bien las puntuaciones cuando se acaba el tiempo?

Si.

¿El jugador que tiene el turno es el único que puede mandar preguntas?

Como ya se ha comentado antes, en un principio no, pero después de bloquear los mandos de los demas jugadores el sistema funciona de forma correcta.

¿Funciona de forma correcta el rebote?

Si, siempre pasa el turno al jugador de la derecha o si es el único jugador al primer jugador. Además si todos han contestado no se pasa el turno a un jugador que ya ha contestado sino que se muestra la respuesta correcta.

¿Se señala acertadamente la respuesta correcta a la pregunta formulada?

Si, tanto si se acierta como si no se acierta.

7.1.11. Incremento 11: Gestión de Records

¿Se crea y destruye objetos de la clase GestionRecords sin problemas?

Si.

¿Se actualizan las puntuaciones de los concursos de forma correcta?

No, hubo problemas a la hora de guardar la información en memoría secundaria, ya que a veces introducia datos erroneos. Era un fallo en el sistema de guardar los datos que se saltaba una de las puntuaciones. Tampoco se actualizaba de forma correcta los nombres cuando el jugador cambia su nombre por otro, se arreglo y ya funciona correctamente.

¿Se borra de forma correcta las puntuaciones de los jugadores que se borran?

Si.

¿Se muestran de forma correcta las puntuaciones en pantalla?

Si.

7.1.12. Incremento 12: Gestión de Jugadores

¿Se crea y destruye objetos de la clase GestionJugadores sin problemas?

Si.

¿Se crea un jugador de forma correcta?

Si.

¿Se puede editar un jugador de forma correcta?

Si.

¿Se muestran los jugadores existentes de forma correcta?

Si.

¿Se borran los jugadores de forma correcta?

No, debido a un fallo de programación sólo se borraba el primer jugador. Una vez arreglado el problema el sistema funciona correctamente.

7.1.13. Incremento 13: Diseño e Implementación del Concurso Pasa la Bomba

¿Se carga el juego de forma correcta?

Si.

¿Se contabilizan bien las puntuaciones al responder una pregunta?

Si.

¿Se cumple el tiempo esperado para el estallido de la bomba?

Si.

¿Se pasa de un turno a otro de forma correcta?

No, cuando a un jugador le explotaba la bomba, la seguía teniendo al siguiente turno. Esto se ha arreglado y ya no ocurre, sino que pasa al siguiente jugador.

¿Se señala acertadamente la respuesta correcta a la pregunta formulada?

Si, tanto si se acierta como si no se acierta.

7.1.14. Incremento 14: Diseño e Implementación del Concurso Gran Concurso

¿Se pueden elegir todos los conursos y en cualquier orden?

No, por un fallo de diseño, sólo se podían escoger los concursos en orden, ya que sino no cargan el concurso que nosotros queriamos, esto se debía a que se utilizaba los indices del vector en donde estaban almanecadas las opciones del menú como id de concurso. Al quitar un concurso del centro de las opciones los indices cambian y entonces no se refieren al id del concurso. Para ello lo que se ha hecho es mirar la frase que tiene almacenada el índice en vez del índice. De esta manera siempre escogemos el concurso que queremos, aunque lo eligamos de forma desordenada.

¿Se guardan los jugadores y su información de un concurso a otro?

Si, no hay problemas a la hora de pasar los vectores de jugadores entre los diferentes concursos.

Capítulo 8

Conclusiones

En esta sección voy a presentar las las conclusiones que he sacado después de haber realizado tanto esfuerzo para realizar el proyecto Prerres.

Para comenzar, debo reconocer que me ha ocupado mucho más tiempo del que esperaba. He perdido mucho tiempo en algunas partes por no saber muy bien como enfocarlas. Aún así creo que el resultado final ha merecido la pena.

El proyecto una vez finalizado, podemos decir que, es un proyecto de buena calidad, en el que se ha intentado realizar un software sencillo, que cumpla con los objetivos con los que se planifico. Para ello se ha optado por una interfaz gráfica muy simple y una interfaz de control aún más simple e intuitiva. Al utilizar mandos para todas las funcionalidades del juego, descentralizamos los controles del teclado y el ratón y hacemos el juego más interactivo para todos.

A esto, hay que añadirle la biblioteca que se ha liberado para poder tener preguntas en formato gift en memoria principal. Con unas pocas de funciones y clases, podemos manejar de forma eficiente, una gran cantidad de preguntas de todos los tipos soportados por GIFT. Esto no habría sido posible si no es gracias al script de perl de Casiano Rodriguez-Leon, como intermediario entre fichero gift y memoria principal.

Con respecto a lo que he aprendido, pienso que un proyecto fin de carrera, sobre todo un proyecto de tipo software, te ayuda a aprender a organizar un código de la magnitud de una gran aplicación, algo que en la carrera no se aprende, ya que se hacen pequeños programas o ejemplos. Esto sirve de ayuda a la hora de realizar otro proyecto de esta magnitud, ya que permite aprender de los fallos y mejorar la organización de ideas. Me hubiese gustado tener mayor conocimiento en el area de ingeniería de software a la hora de afrontar este proyecto, así me hubiese ahorrado bastante tiempo solucionando problemas de planificación y diseño. Una vez que he finalizado el proyecto, puedo afirmar que ahora creo que puedo afrontar un proyecto con totales garantías de éxito.

Destacar como positivo, la ayuda que puede prestarte las personas de tu entorno. Intentar adecuarlas a tu idea de proyecto y poder acatar los diferentes consejos que te dan. Esto ayuda a detectar fallos que nunca habría detectado si hubiese desarrollado el juego sin opiniones externas. Además hace que el proyecto esté más adaptado a los usuarios que pueden llegar a utilizarlo.

La base del proyecto han sido las distintas librerías que forman *libSDL*. Aunque algunas librerías como la TTF todavían deben mejorar, creo que es una librería muy completa para el desarrollo de pequeñas videojuegos en 2D. Además gracias a tutoriales como el aportado por *Antonio García Alba*, el **Tutorial**

Wiki libSDL [6], hace que sea bastante fácil la familiarización con está biblioteca y conocer las distintas librerías que la componen.

Destacar también el programa de animación *Sinfyg*. Un programa de esa calidad y que sea libre es una gran noticia para todos. Ha sido muy fácil realizar animaciones para luego utilizarlas en el proyecto. Evidentemente cuanto mejor seas diseñando graficamente, mejores resultados dará el programa. Además gracias al taller impartido por Carlos López [7] y a su ayuda, ha sido mucho más fácil el dominio del programa en poco tiempo.

Gracias a herramientas como *Planner* con sus diagramas de Gantt es más fácil e intuitivo ver las tareas que tienes pendiente y el tiempo estimado que dispones para realizarlas. Se puede aprender también que cualquier imprevisto que pueda ocasionar no es una pérdida de tiempo, si no una oportunidad para ver que las cosas pueden hacerse mejor, y que desarrollar con prisas te puede ocasionar que aparezcan numerosos bugs que al final te harán perder más tiempo todavía.

No soy partidario de utilizar entornos de desarrollo por lo que mi proyecto está escrito mediante el editor de texto gedit. Aunque es un editor de texto sencillo, a mi me resulta muy fácil de usar, tiene lo básico para programar en C++ y me ahorra documentarme sobre un entorno de desarrollo y su funcionamiento.

En definitiva, este proyecto me ha hecho madurar como estudiante. He aprendido a consultar bibliografía, a buscar opiniones en compañeros, profesores y amigos, compartir ideas, aprender un poco más de programación y sobre todo a enfrentarme a un proyecto de estas características.

8.1. Posibles Ampliaciones

En futuras versiones el proyecto podría tener una expansión bastante importante, algunas de las ideas que se podía implantar serían:

1. Reconocimiento por voz. Podría implantarse un sistema de reconocimiento de voz para, con ayuda de un micrófono, se pudiera respondera las distintas preguntas mediante la voz.

2. Diferentes controles. Se podría implementar un nuevo sistema de control, mediante gamepad.

3. Preparación de examenes. Podría realizarse un concurso que fuera exclusivamente preguntas de un tema seleccionado y que ayudará a preparar una asignatura en concreto. Se podrían repetir las preguntas que hubiesen sido erroneas, hasta que el usuario las conteste correctamente.

4. Creación de repositorio de lotes de preguntas. Podría realizarse un repositorio en el que cada uno pudiese subir su lote de preguntas para poderlo compartir con los demás jugadores.

5. Nuevos personajes. Se podrían realizar nuevas animaciones de personajes para que hubiese más variedad de elección.

6. Mejorar gráficos. Se podría adaptar mejor los gráficos y hacerlo de un modo más profesional.

Apéndice A

Manual de Usuario

A continuación detallamos las instrucciones básicas que un usuario normal debe saber sobre el proyecto Prerres.

A.1. Ejecución

Es posible la ejecución del programa en cualquier entorno GNU/Linux. Una vez finalizado la instalación del producto simplemente tendremos que escribir en una consola:

```
./prerres
```

También es posible ejecutarlo mediante doble click en entorno gráfico aunque se recomienda en consola para tener más información sobre la ejecución.
Una vez ejecutado el comando anterior veremos el menú principal de la aplicación.

A.2. Primeros Pasos

Para el control de la aplicación se recomienda utilizar los Buzzers de PS2 aunque se puede utilizar otro tipo de joystick siempre adecuados a la cantidad de jugadores que vayan a jugar. Es decir, 1 joystick 1 jugador, 2 joystick 2 jugadores, etc... Además se puede manejar por teclado.
Al ejecutar Prerres, se muestra el menú principal. Esta es simple como puede verse en la figura siguiente, y nos dá acceso a los Concursos, a las Opciones, Gestión de Jugadores, Gestión de Records y Salir de la aplicación.

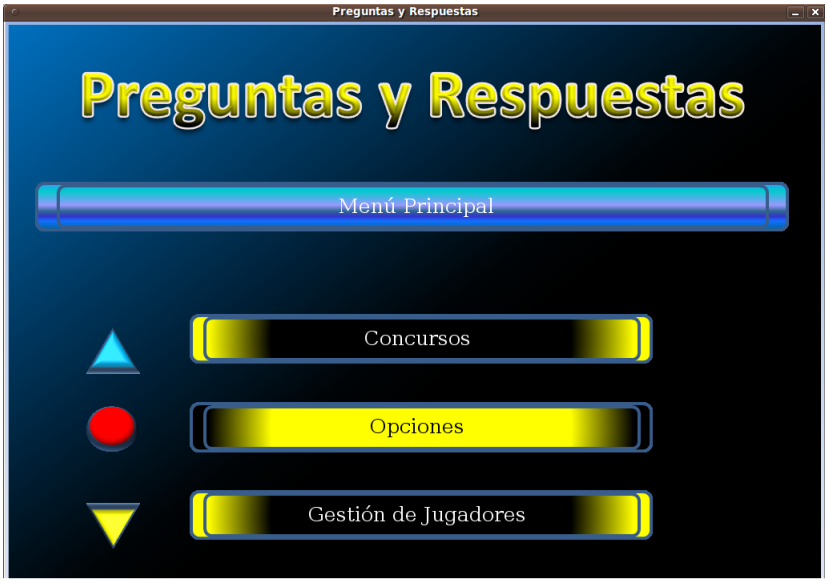

Figura A.1: Menú principal de Prerres

Para empezar a jugar habrá que crearse mínimo un jugador en el área de Gestión de Jugadores. Una vez hecho esto, podemos empezar por algún concurso sencillo como el Normal o Contrarreloj.

A.3. Controles

A.3.1. Joystick

Está es la manera óptima de disfrutar del juego. Gráficamente el juego está diseñado para utilizar los Buzzers de PS2, aunque se podría utilizar otro tipo de joystick (poco recomendado). Las funcionalidad de los distintos botones del joystick se indican a continuación:

- Botón Rojo: Normalmente sirve de confirmación en cualquier menú del juego. No se utiliza en el desarrollo de los concursos.
- Botones de colores: Estos sirven tanto para moverse por los menús como para responder a las preguntas en los concursos. En cada momento, se indicará en pantalla la función de cada uno de ellos. Si no aparece en pantalla, es que no tiene funcionalidad en esa área.

Figura A.2: Funcionalidad del Mando

A.3.2. Teclado

También se da la opción de la utilización del teclado, aunque no es la más recomendada para jugar varias personas. Los controles serían los siguientes:

- Tecla Intro: Haría las veces de botón rojo. Es la tecla de confirmación.
- Para cada jugador se dispone de un conjunto de letras que simula los botones de colores:
  - Jugador 1: Botón Azul: Q, Botón Naranja: W, Botón Verde: A, Botón Amarillo: S.
  - Jugador 2: Botón Azul: R, Botón Naranja: T, Botón Verde: F, Botón Amarillo: G.
  - Jugador 3: Botón Azul: U, Botón Naranja: I, Botón Verde: J, Botón Amarillo: K.
  - Jugador 4: Botón Azul: 1, Botón Naranja: 2, Botón Verde: 3, Botón Amarillo: 4. Del teclado número.

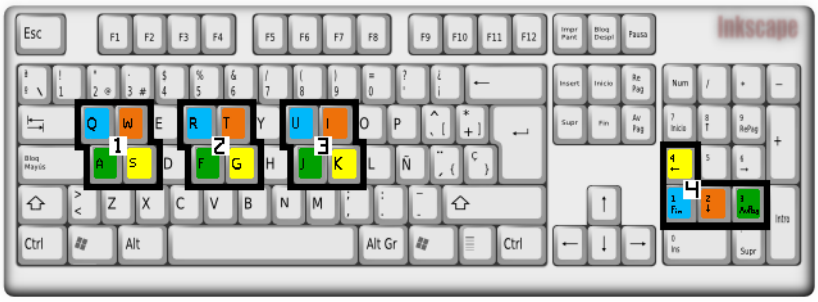

Figura A.3: Funcionalidad del Teclado

A.4. Modos de Juego

Si escogemos la opción Concursos nos preguntará el número de jugadores que van a jugar. Una vez elegido se nos presentan los concursos a los que podemos jugar con ese número de jugadores. A iniciar un concurso por defecto se cogen las opciones por defecto del jugador 1, pero se pueden cambiar antes de empezar el concurso desde el menú opciones que nos aparece.

Figura A.4: Menú de elección del número de jugadores

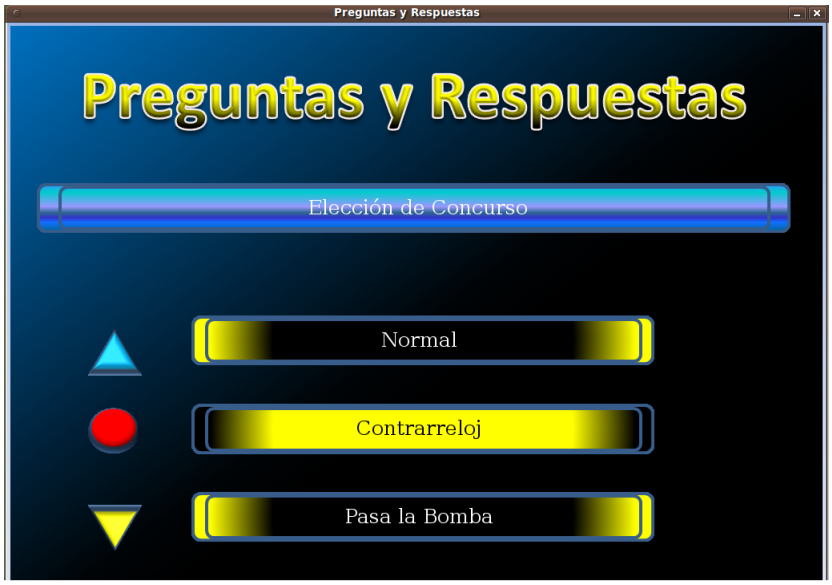

Figura A.5: Menú de concursos

Figura A.6: Menú concurso Normal

Seguidamente se explica la pantalla de juego en la siguiente figura.
Iconos:

-  Indica los puntos conseguidos por el usuario.
-  Indica el tiempo que le queda al usuario para responder.

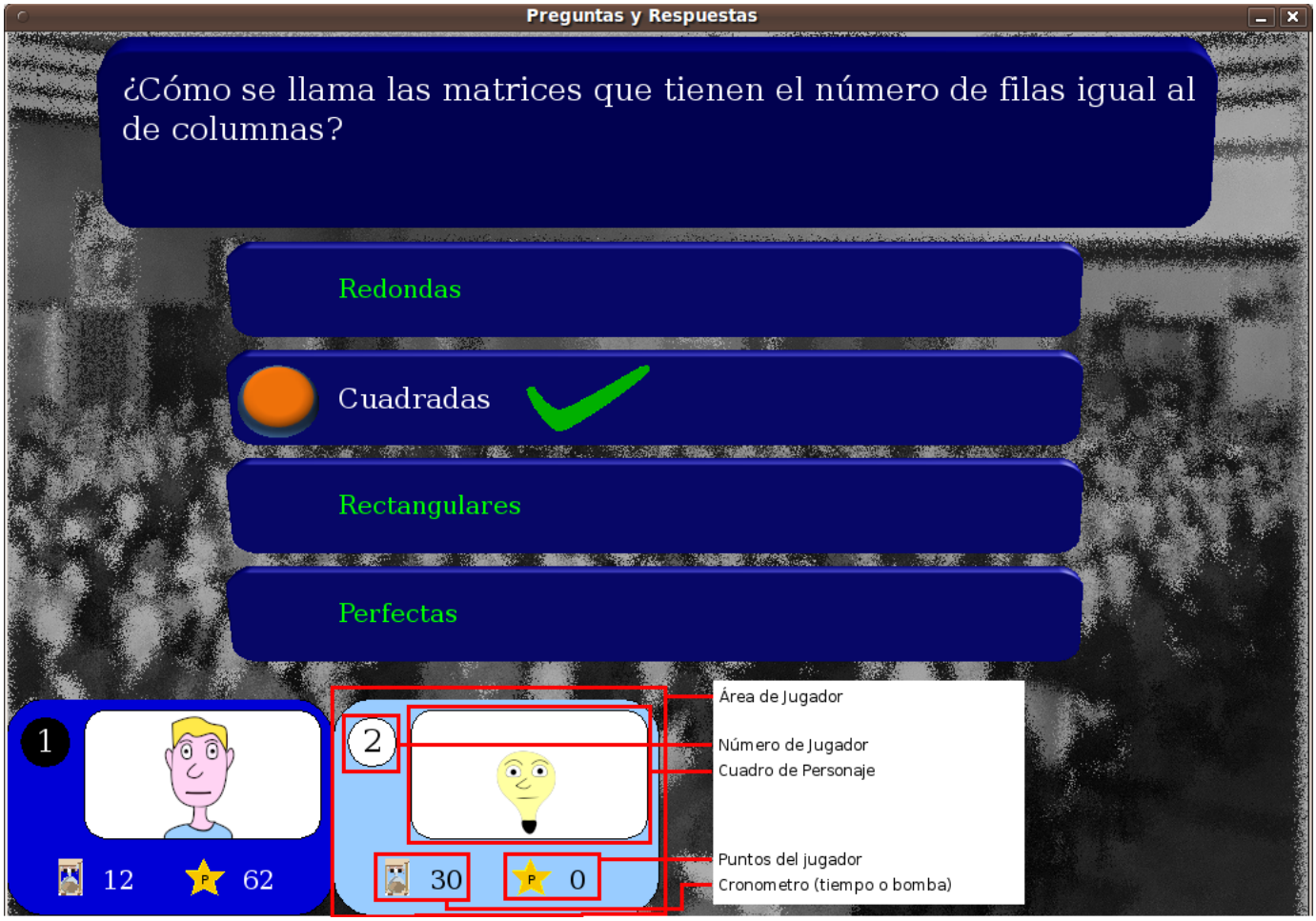

Figura A.7: Descripción de la pantalla de juego

A continuación se explica cada uno de los concursos que existen en Prerres.

A.4.1. Concurso Normal

- Jugadores: 1-4
- Trofeo: 
- Funcionamiento: Concurso por turnos. Se muestra una pregunta al jugador y debe contestarla en el tiempo que se le proporciona.
- Puntuación: Si falla la pregunta o se le acaba el tiempo: -50 puntos. Si acierta la pregunta: Depende del tiempo de contestación 100 o menos puntos. En cualquier caso pasa el turno al jugador de su derecha.

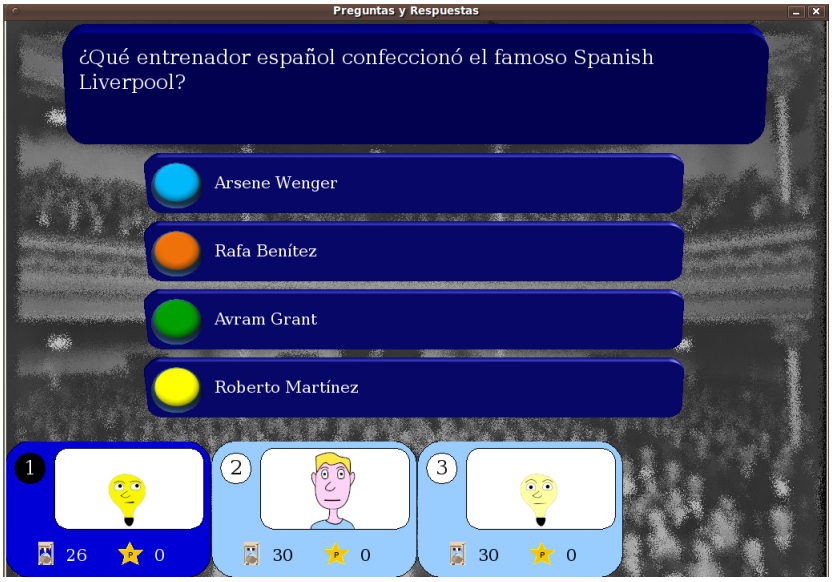

Figura A.8: Pantalla del concurso Normal

A.4.2. Concurso Contrarreloj

- Jugadores: 1-4
- Trofeo: 
- Funcionamiento: Concurso por turnos. Se muestra una pregunta al jugador activo y este debe responder en el menor tiempo posible. Cuantó más rapido lo haga más puntos podrá conseguir.
- Puntuación: Si se acierta la pregunta: Depende del tiempo de contestación 100 o menos puntos. Si se falla la pregunta: -50 puntos.

Figura A.9: Pantalla del concurso Contrarreloj

A.4.3. Concurso Pasa la Bomba

- Jugadores: 2-4
- Trofeo: 
- Funcionamiento: Concurso de agilidad. Se muestra una pregunta y el jugador debe responder correctamente antes de que le explote la bomba.
- Puntuación: Por contestar correctamente y pasar la bomba: Depende del tiempo de contestación 100 o menos puntos. Si se falla la pregunta: 0 puntos. Si explota la bomba: -50 puntos

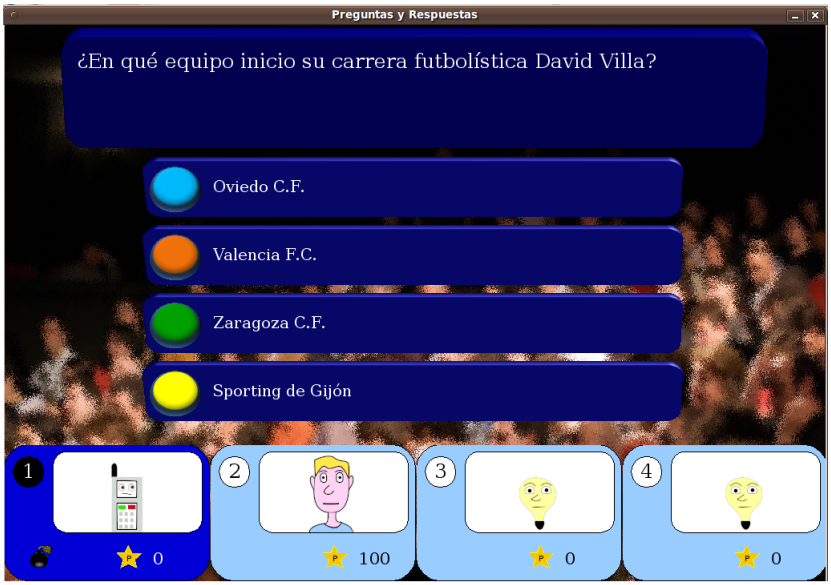

Figura A.10: Pantalla del concurso Pasa la Bomba

A.4.4. Concurso Tonto el Último

- Jugadores: 2-4
- Trofeo: 
- Funcionamiento: Concurso por pulsación. Se muestra una pregunta a los jugadores y deben contestarla en el tiempo que se le proporciona, pudiendo cambiar su respuesta cada vez que quieran.
- Puntuación: Si ha sido el primero en contestar la pregunta y es correcta: +100 puntos. Si se selecciona la respuesta correcta pero alguién ha contestado antes: 0 puntos. En cualquier otro caso: -50 puntos.

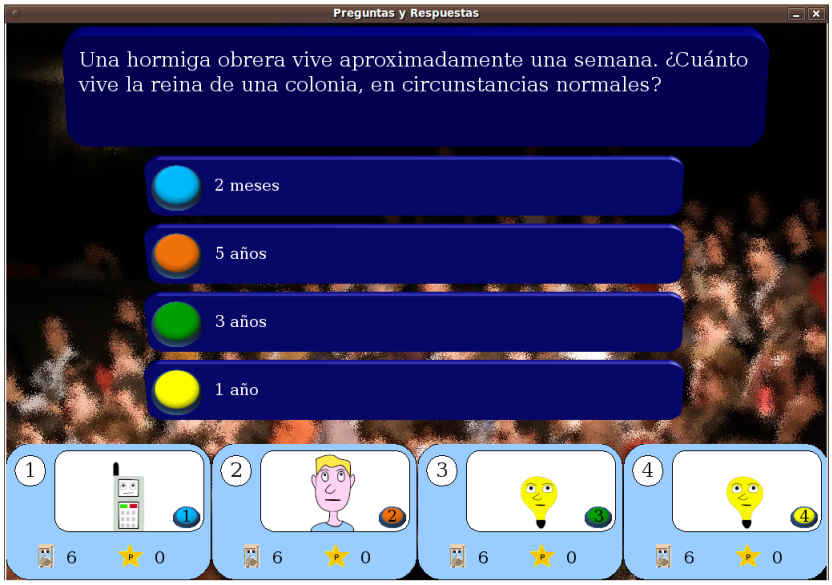

Figura A.11: Pantalla del concurso Tonto del último

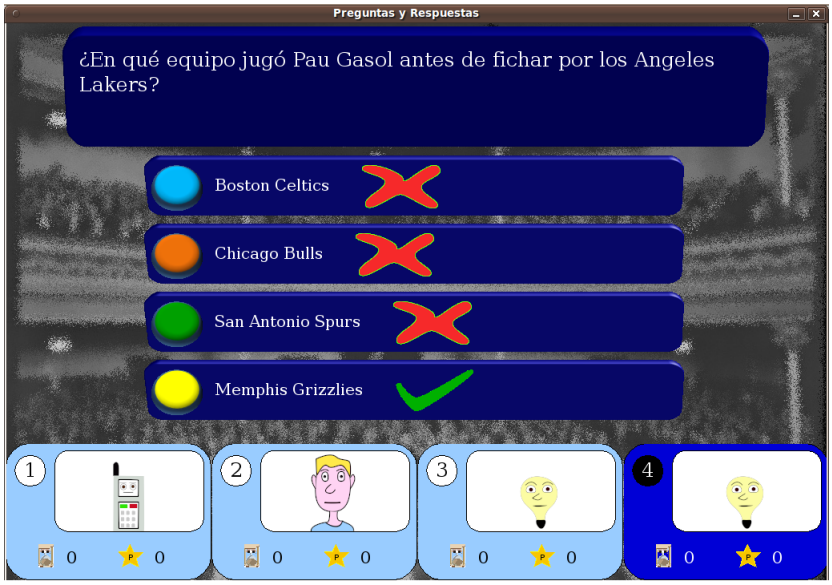

Figura A.12: Comprobación de las respuestas de los usuarios en el concurso Tonto del último

A.4.5. Concurso Lanza la Pregunta

- Jugadores: 2-4
- Trofeo: 
- Funcionamiento: Concurso por turnos. Se muestra una pregunta a los jugadores y el jugador que tiene el turno elige a que otro jugador lanza la pregunta. Una vez que lo elige el jugador debe

contesta contestarla en el tiempo que se le proporciona. Si el jugador falla la pregunta, el turno pasa al jugador de su derecha hasta que se acierta o no hay más jugadores.

- Puntuación: Si se acierta la pregunta: Depende del tiempo de contestación 100 o menos puntos. Si se falla la pregunta: -50 puntos.

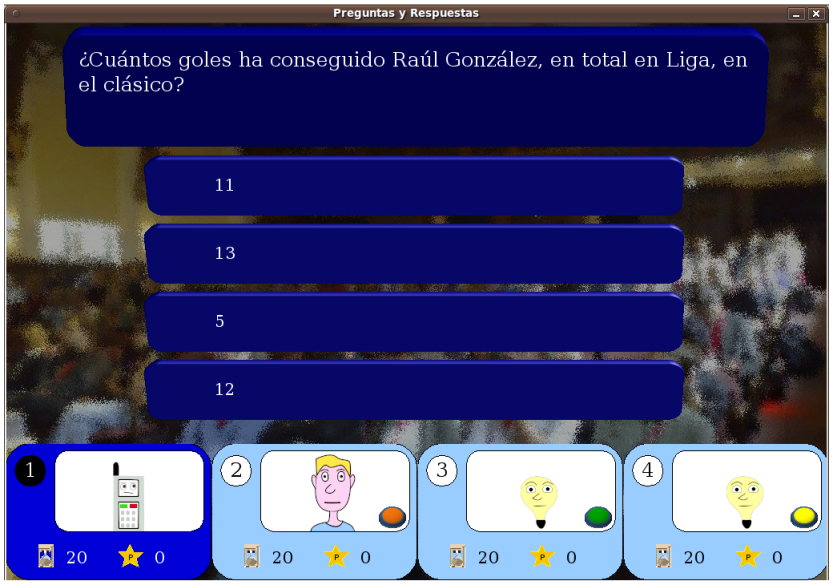

Figura A.13: Pantalla del concurso Lanza la pregunta

A.4.6. Concurso Gran Concurso

- Jugadores: 3-4
- Trofeo: 
- Funcionamiento: Este concursos simplemente es un remix de los demás. Se pueden encadenar todos los concursos anteriores. Finalmente se mostrará la puntuación final de todos los juegos.
- Puntuación: La de cada concurso.

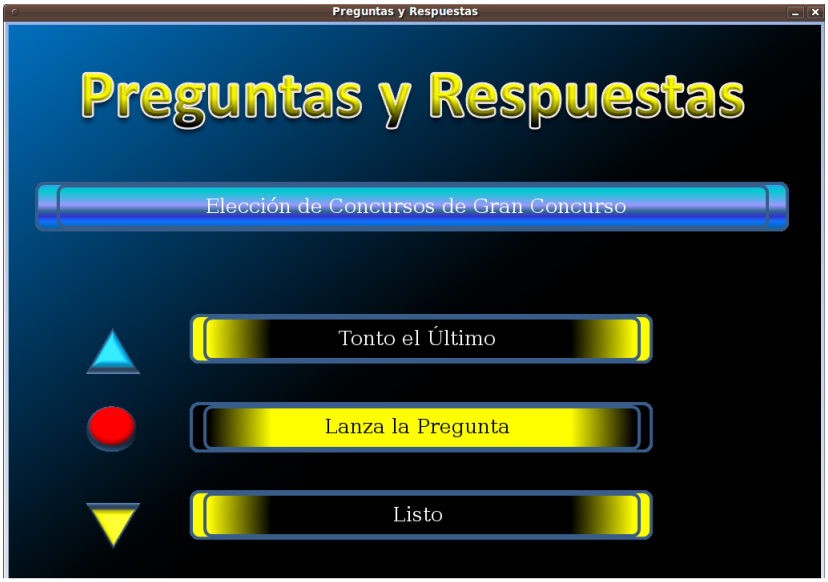

Figura A.14: Pantalla del concurso Gran Concurso

A.4.7. Opciones del Juego

Si ha optado por acceder al menú opciones le aparecerá una pantalla como la siguiente:

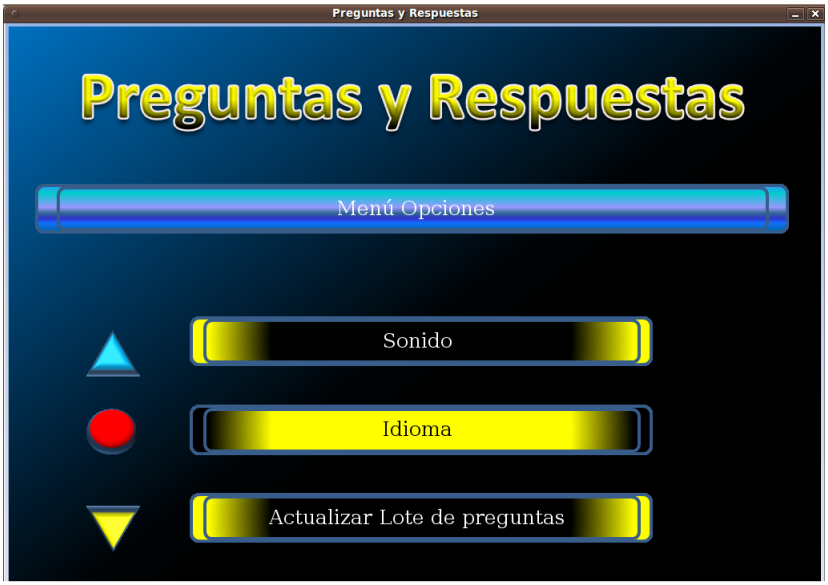

Figura A.15: Menú de Opciones de Prerres

Podemos acceder a las siguientes opciones que se describen a continuación:

- Sonido: Activa o Desactiva el sonido de forma predeterminada.

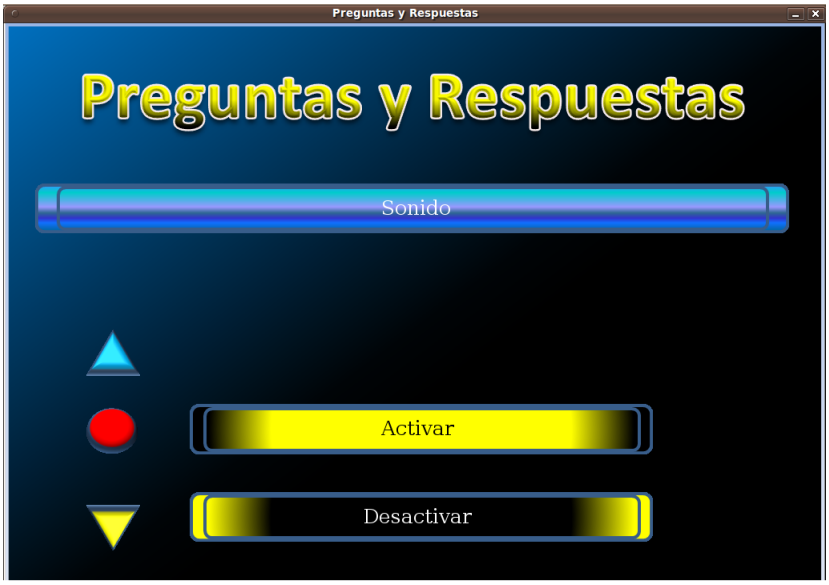

Figura A.16: Opciones de sonido

- Idioma: Selecciona Español o Inglés de forma predeterminada.

Figura A.17: Opciones de idioma

- Actualizar Lote de Preguntas: Si se han cambiado el fichero gift de preguntas durante la ejecución del juego es posible cargarlo mediante esta opción.

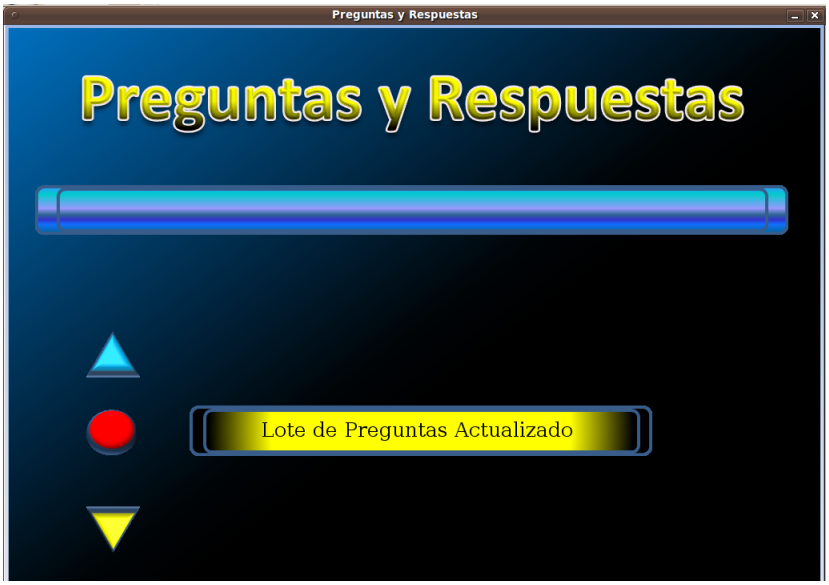

Figura A.18: Actualización de las preguntas.

Para regresar al menú principal simplemente hay que escoger la opción Volver.

A.5. Gestión de Jugadores

La gestión de jugadores nos permite estás opciones:

- Nuevo Jugador.
- Editar Jugador.
- Mostrar Jugador.
- Borrar Jugador.

Figura A.19: Gestión de Jugadores.

A continuación se comentan brevemente cada una de las secciones.

A.5.1. Nuevo Jugador

Si queremos crear un jugador, deberemos acceder a esta función. Nos pedirá un nombre, un personaje, categoría por defecto, el número de preguntas/bombas por defecto y el tiempo de contrarreloj por defecto.

Figura A.20: Introduciendo un nombre.

Figura A.21: Escogiendo animación.

Figura A.22: Eligiendo categoría de preguntas.

Figura A.23: Eligiendo categoría de preguntas.

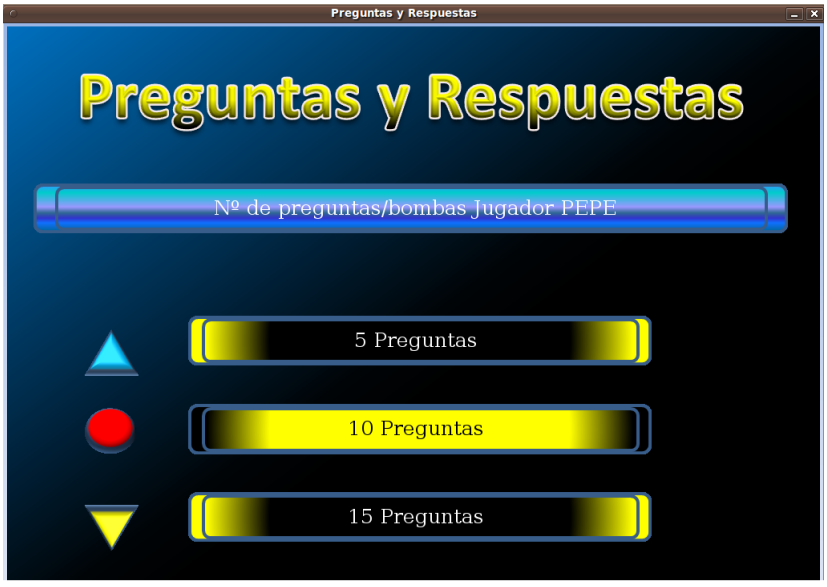

Figura A.24: Eligiendo categoría de preguntas.

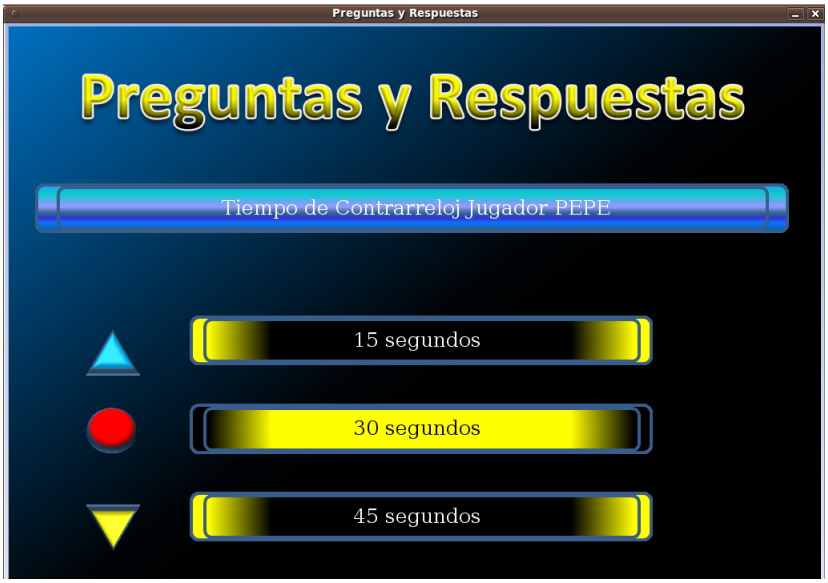

Figura A.25: Eligiendo tiempo de contrarreloj.

A.5.2. Editar Jugador

En esa sección podemos modificar todas las características que elegimos al crear al jugador. Podemos cambiar el nombre, personaje, categoría por defecto, el número de preguntas/bombas por defecto y el tiempo de contrarreloj por defecto.

Figura A.26: Menú de editar jugador.

A.5.3. Mostrar Jugador

En está sección podemos ver tanto las características elegidas en la creación del jugador como algunas estadísticas sobre su participación en concurso.

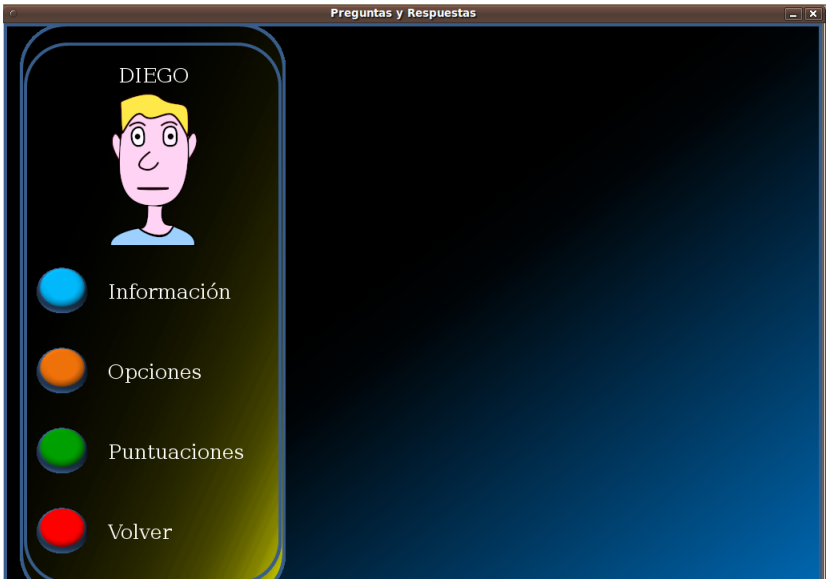

Figura A.27: Pantalla de mostrar jugador

En concreto está dividido en tres secciones:

- Información: Número de preguntas respondidas y rating de contestación.
- Opciones: Opciones de concursos por defecto.
- Puntuaciones: Mejor puntuación de cada concurso.

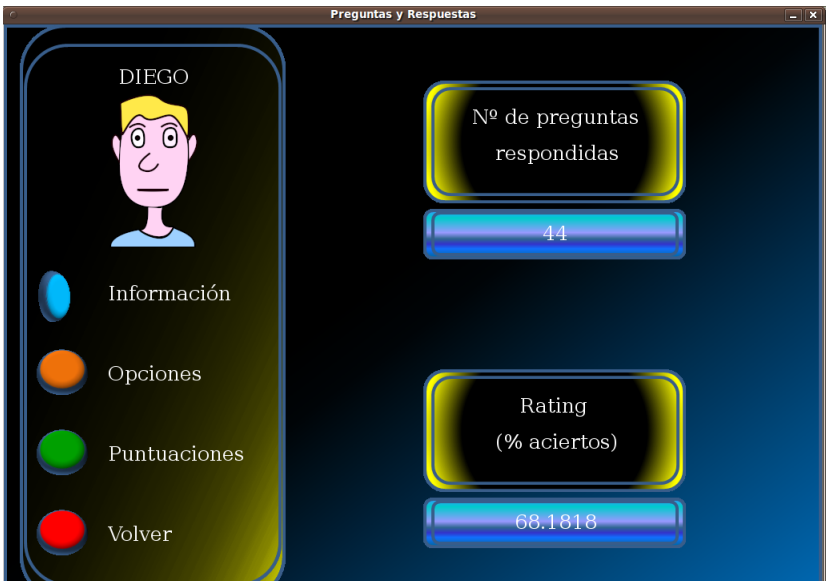

Figura A.28: Información de jugador

A.5.4. Borrar Jugador

También podemos borrar un jugador. Esto conlleva borrar tanto su perfil, como sus puntuaciones generales en los concursos. Así que hay que tener cuidado a la hora de borrar un jugador. Por eso el sistema nos pregunta si queremos borrar el jugador antes de hacerlo.

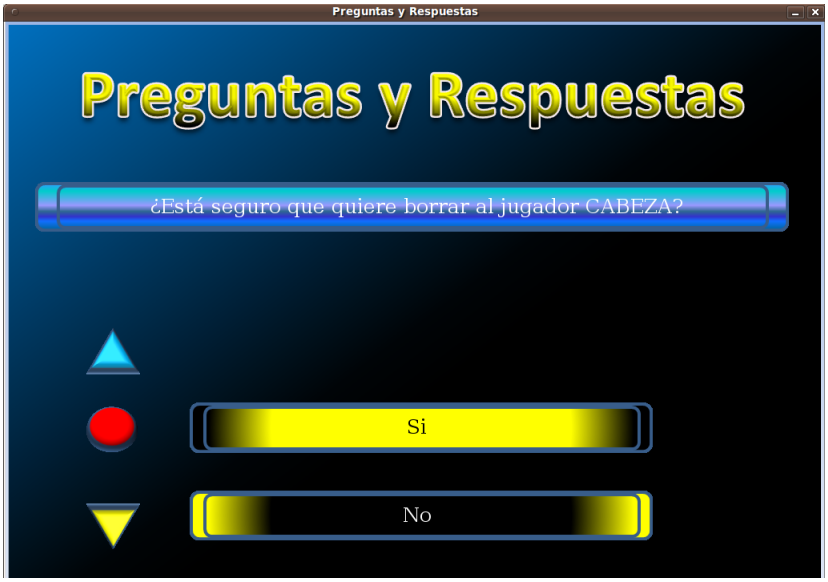

Figura A.29: Borrando un jugador

A.6. Records

En este área podemos consultar las mejores puntuaciones de los concursos ordenados por categoría. Elegimos una categoría y se nos muestran las puntuaciones obtenidas hasta el momento en cada concurso en esa categoría.

Figura A.30: Ejemplo de Puntuaciones

A.7. Categorías

De forma predeterminada hay 4 categorías:

- General: Preguntas de distintos temás.
- Historia: Preguntas sobre historia.
- Ciencias: Preguntas sobre matemáticas, biología, física, etc.
- Deportes: Preguntas sobre fútbol, tenis, baloncesto, etc.

¡Pero esto no es fijo! Al introducir tus propias preguntas, ¡también puedes introducir tus propias categorías!. Esto se explica en el Apéndice C.

Figura A.31: Captura del menú de elección de categorías para los Records

Apéndice B

Manual de Instalación

B.1. Obtener Prerres

En primer lugar deberemos obtener una versión de Prerres. Lo podemos encontrar en la página del proyecto en la forja de rediris [8] o haciendo un chekout mediante subversion de la siguiente manera:

```
svn checkout https://forja.rediris.es/svn/prerres
```

B.2. Instalación

Si hemos optado por bajarnos el paquete comprimido de la página del proyecto, primero deberemos descomprimirlo, podremos hacerlo fácilmente haciendo:

```
tar -xvzf prerres-v1.0.tar.gz
```

Una vez que hemos descomprimido el juego o hemos optado por usar subversion, debemos instalarnos el script de Gift-0.6. Para ello nos situamos en la carpeta src/gift y seguimos las instrucciones del fichero REEDME.
Si no tenemos instaladas las librerías SDL, tendremos que instalarlas. En este caso basta con instalar la librería y las librerías adicionales mixer, image y tff mediante su gestor de paquetes o la línea de comandos. En un entorno debian los comandos a ejecutar serían:

```
sudo atp-get install libsdl1.2debian
sudo atp-get install libsdl-ttf2.0-0-dev
sudo atp-get install libsdl-image1.2-dev
sudo atp-get install libsdl-mixer1.2-dev
```

Seguidamente nos situamos en la carpeta src y compilamos el proyecto.

```
cd prerres/src/
```

Luego para compilar simplemente haremos “make”.

```
make
```

Si la compilación ha tenido éxito nos generará un fichero ejecutable llamado prerres, por tanto sólo tendremos que ejecutarlo.

```
./prerres
```

Apéndice C

Manual para agregar nuevas preguntas y categorías

En este apartado mostraremos como introducir nuevas preguntas al fichero general de preguntas. Es importante que se añadan al final de este y que no se modifique el contenido que ya tiene, ya que podría ocasionar que alguna de las preguntas que ya están introducidas no funcionen debidamente.

C.1. Introduciendo nuevas preguntas

A continuación se va a explicar paso a paso como introducir nuevas preguntas del tipo multiopción al fichero general de preguntas. Para empezar se muestra la sintaxis básica de las preguntas multiopción para el formato GIFT.

[Categoria] Pregunta {
    = Respuesta Correcta
    ~Respuesta Incorrecta
} [1]

Es importante dejar un salto de línea tanto al comenzar la pregunta como al terminarla, sino las preguntas no se guardarán de forma correcta.

Como se puede ver, entre corchetes podemos poner la categoría[2]. La sintáxis es sensible a capitalización. Por lo tanto, si se introduce una pregunta de una categoría ya existente, hay que escribirla tal cúal está establecida en las demás preguntas de la categoría. Además no puede contener carácteres con acentos o símbolos extraños, unicamente los recogidos en la tabla ASCII [11].

Entre el último corchete y la apertura de llave se introduce la pregunta. No es posible introducir salto de línea.

Dentro de las llaves se pueden agregar las respuestas que se desee, siempre que sólo haya una correcta.

[1] Aunque este formato permite pesos, comentarios y nombres de preguntas, al no utilizarlo el juego, es contraproducente explicarlo a la hora de enseñar la sintáxis básica para agregar una pregunta. Para más información sobre el formato GIFT consultar la bibliografía [9] ó [10].

[2] En la sintáxis del formato GIFT, la opción entre corchetes significa modo de visualización. Al no tener sentido a la hora de pasarlo a memoria principal, se reutiliza para indicar la categoría de las preguntas

Ejemplo:
[General]¿De qué región es la paella?{
~Cataluña
~Galicia
~Bilbao
=Valencia
~Andalucía
}
En este ejemplo, la categoría de la pregunta sería *General*, la pregunta sería *¿De qué región es la paella?*, las respuesta incorrectas *Cataluña, Galicia, Bilbao, Andalucía* y la correcta *Valencia*.

Como se ve, hay 5 respuestas para esa pregunta. En el juego sólo se verán 4 de ellas, entre las que siempre estará la correcta y 3 incorrectas escogidas al azar entre las que hubiese. Cuantas más respuestas se añadan más diferentes serán las preguntas.

C.2. Manipulando el fichero Preguntas.txt

Este fichero es el que contiene todas las preguntas del juego. Estará situado en la carpeta *./prerres/src/gift/script* y sólo lo manipularemos si queremos modificar las preguntas del juego.

Las preguntas se añaden al final del fichero o en su defecto, detrás de la última pregunta de la categoría que se quiera insertar la pregunta, con el formato ya mencionado anteriormente.

Una vez terminada la edición del fichero se guardará con el mismo nombre y en la misma carpeta. Si el juego está ejecutandose se puede actualizar las nuevas preguntas con la opción *Actualizar Lote de Preguntas* A.18. En caso de no estar ejecutandose, en la próxima ejecución las nuevas preguntas estarán incluidas automáticamente.

C.3. Agregar una nueva categoría

Para agregar una nueva categoría al lote de preguntas, sólo hay que crear una pregunta que sea de esa nueva categoría. Esta categoría debe ser una cadena ASCII y desde ese momento, las nuevas preguntas que se quieran añadir de esa categoría tendrán que tener la misma capitalización que la primera que se añadió.

Apéndice D

Biblioteca de preguntas con formato GIFT

En este anexo se va a detallar en que consiste exactamente la Biblioteca de preguntas con formato GIFT que se va a liberar con este proyecto.

En primer lugar vamos a introducir el formato GIFT. El formato GIFT permite usar un editor de texto para escribir preguntas de opción múltiple, verdadero-falso, respuesta corta, incluir palabra que falta y preguntas numéricas en un formato simple que puede ser importado. Para más información consultar la documentación de Moodle [9].

Este tipo de formato es posible utilizarlo para la plataforma moodle y está adaptado para tal fin. Yo lo he reutilizado y he creado una biblioteca que pasandole un fichero en formato GIFT, sin la opción $CATEGORY y readaptando la opción de formato de visual por la Categoría de las preguntas, pueda pasar las preguntas contenidas en él a memoria principal en forma de clases.

Es importante hacer saber, que para utilizar esta biblioteca debemos pasar antes el script de perl que se adjunta con la biblioteca al fichero que queramos convertir, ya que la clase Lote, lo que hace es recoger las preguntas de el fichero que devuelve dicho script.

Primeramente vamos a presentar las clases en las que realmente están recogidas las preguntas del fichero y seguidamente presentaremos lo que puede llamarse el contenedor de todas esas preguntas.

D.1. Clase Pregunta y sus hijas

Empezamos presentamos el conjunto de clases que recoge los diferentes tipos de preguntas en formato GIFT. La clase Pregunta es una generalización de las distintas clases que recogen los distintos tipos de preguntas con las características comúnes de todas ellas. Así podemos encontrar la clase MultiOpcion, que se corresponde con las preguntas de tipo opción múltiple, la clase Corta, que se corresponden con las preguntas de respuesta corta, etc.

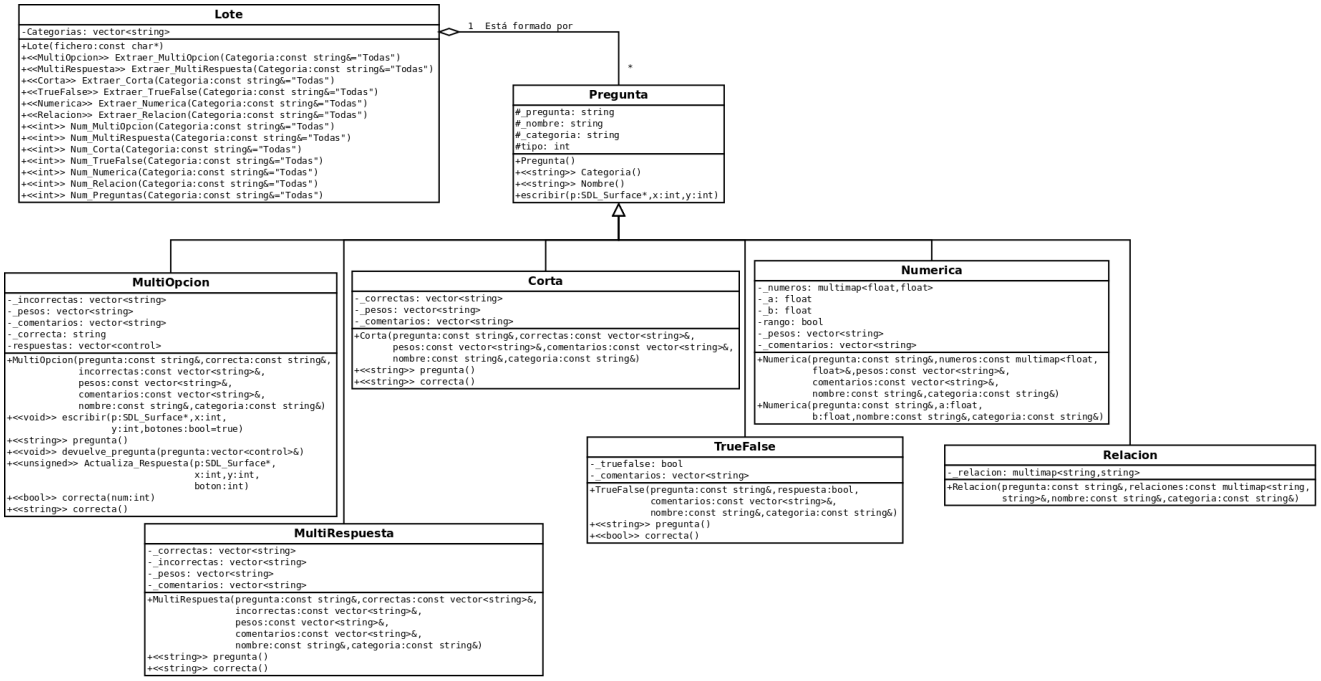

Figura D.1: Diagrama de clases de la Clase Pregunta

Cada una de las clases hijas se presentan a continuación:

D.1.1. Clase MultiOpcion

Presentamos esta clase que es la que recoge las preguntas de opción múltiple del formato GIFT. Esta es la clase que se utiliza para el videojuego.

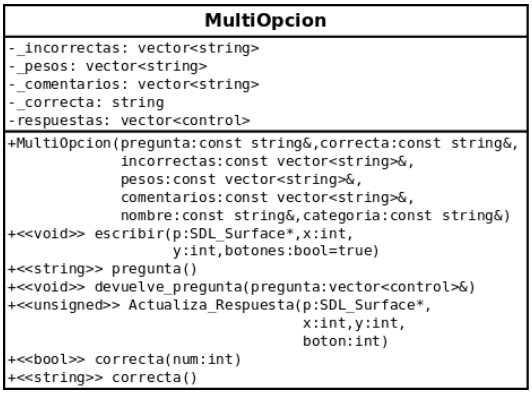

Figura D.2: Clase MultiOpcion

Basicamente consta:

- **Constructor: MultiOpcion(const string& pregunta, const string& correcta, const vector<string>& incorrectas, const vector<string>& pesos, const vector<string>& comentarios, const string& nombre, const string& categoria)**. Construye un objeto de la clase MultiOpcion o lo que es lo

mismo, una pregunta del tipo opción múltiple. Donde *pregunta* es el texto de la pregunta, *correcta* es la respuesta correcta, *incorrectas* es un vector de respuestas incorrectas, *pesos* es un vector con los pesos para cada respuesta, *comentarios* es un vector con los comentarios a cada respuesta, *nombre* es el nombre que se le asigna a la pregunta y *categoria* es la categoría a la que pertenece la pregunta.

- **Constructor: MultiOpcion(const MultiOpcion& mo)**. Constructor de copia de la clase MultiOpcion.
- **Destructor: ~MultiOpcion()**. Libera los recursos de un objeto de la clase MultiOpcion.
- **Operador: MultiOpcion& operator =(const MultiOpcion& mo)**. Operador de asignación para la clase MultiOpcion.
- **Consultora: string Categoria()**. Esta función es heredada de la clase Pregunta y nos devuelve la Categoría a la que corresponde la pregunta en forma de string.
- **Consultora: string Nombre()**. Esta función es heredada de la clase Pregunta y nos devuelve la Nombre asignado a la pregunta en forma de string.
- **Consultora: string pregunta()**. Devuelve el texto de la pregunta en forma de string.
- **Consultora: bool correcta(int num)**. Devuelve si la respuesta número *num* es la respuesta correcta.
- **Consultora: string correcta()**. Devuelve el texto de la respuesta correcta en forma de string.

D.1.2. Clase MultiRespuesta

Presentamos esta clase que es la que recoge las preguntas de opción múltiple en la que puede haber más de una respuesta correcta.

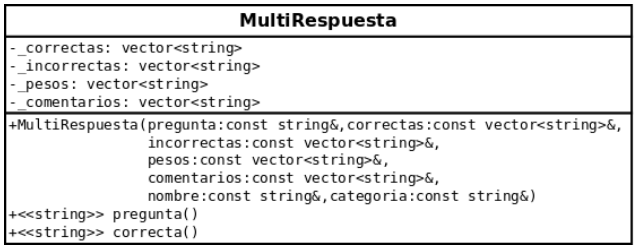

Figura D.3: Clase MultiRespuesta

Basicamente consta:

- **Constructor: MultiRespuesta(const string& pregunta, const vector<string>& correctas, const vector<string>& incorrectas, const vector<string>& pesos, const vector<string>& comentarios, const string& nombre, const string& categoria)**. Construye un objeto de la clase MultiRespuesta. Donde *pregunta* es el texto de la pregunta, *correctas* es un vector con las repsuestas correctas, *incorrectas* es un vector de respuestas incorrectas, *pesos* es un vector con los pesos para cada respuesta, *comentarios* es un vector con los comentarios a cada respuesta, *nombre* es el nombre que se le asigna a la pregunta y *categoria* es la categoría a la que pertenece la pregunta.

- **Constructor: MultiRespuesta(const MultiRespuesta& mr)**. Constructor de copia de la clase MultiRespuesta.
- **Destructor: ~MultiRespuesta()**. Libera los recursos de un objeto de la clase MultiRespuesta.
- **Operador: MultiRespuesta& operator =(const MultiRespuesta& mr)**. Operador de asignación para la clase MultiRespuesta.
- **Consultora: string Categoria()**. Esta función es heredada de la clase Pregunta y nos devuelve la Categoría a la que corresponde la pregunta en forma de string.
- **Consultora: string Nombre()**. Esta función es heredada de la clase Pregunta y nos devuelve la Nombre asignado a la pregunta en forma de string.
- **Consultora: string pregunta()**. Devuelve el texto de la pregunta en forma de string.
- **Consultora: string correcta()**. Devuelve el texto de la respuesta correcta en forma de string.

D.1.3. Clase Corta

Presentamos esta clase que es la que recoge las preguntas de respuesta corta del formato GIFT.

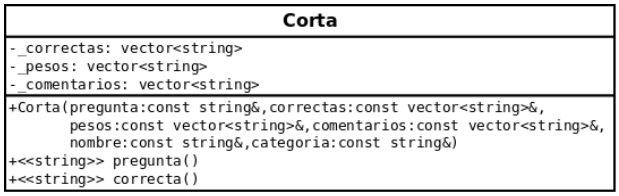

Figura D.4: Clase Corta

Basicamente consta:

- **Constructor: Corta(const string& pregunta, const vector<string>& correctas, const vector<string>& pesos, const vector<string>& comentarios, const string& nombre, const string& categoria)**. Construye un objeto de la clase Corta o lo que es lo mismo, una pregunta del tipo respuesta corta. Donde *pregunta* es el texto de la pregunta, *correctas* es un vector con las repsuestas correctas, *pesos* es un vector con los pesos para cada respuesta, *comentarios* es un vector con los comentarios a cada respuesta, *nombre* es el nombre que se le asigna a la pregunta y *categoria* es la categoría a la que pertenece la pregunta.
- **Constructor: Corta(const Corta& c)**. Constructor de copia de la clase Corta.
- **Destructor: ~Corta()**. Libera los recursos de un objeto de la clase Corta.
- **Operador: Corta& operator =(const Corta& c)**. Operador de asignación para la clase Corta.
- **Consultora: string Categoria()**. Esta función es heredada de la clase Pregunta y nos devuelve la Categoría a la que corresponde la pregunta en forma de string.
- **Consultora: string Nombre()**. Esta función es heredada de la clase Pregunta y nos devuelve la Nombre asignado a la pregunta en forma de string.
- **Consultora: string pregunta()**. Devuelve el texto de la pregunta en forma de string.
- **Consultora: string correcta()**. Devuelve el texto de la respuesta correcta en forma de string.

D.1.4. Clase TrueFalse

Presentamos esta clase que es la que recoge las preguntas de verdadero-falso del formato GIFT.

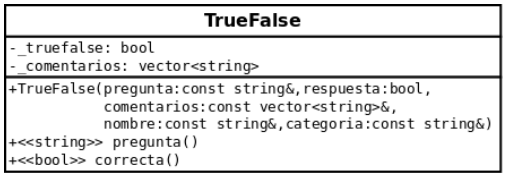

Figura D.5: Clase TrueFalse

Basicamente consta:

- **Constructor: TrueFalse(const string& pregunta, bool respuesta, const vector<string>& comentarios, const string& nombre, const string& categoria)**. Construye un objeto de la clase TrueFalse o lo que es lo mismo, una pregunta del tipo verdadero-falso. Donde *pregunta* es la afirmación, *respuesta* indica si es verdad o mentira dicha afirmación,*comentarios* es un vector con los comentarios a cada respuesta, *nombre* es el nombre que se le asigna a la pregunta y *categoria* es la categoría a la que pertenece la pregunta.
- **Constructor: TrueFalse(const TrueFalse& c)**. Constructor de copia de la clase TrueFalse.
- **Destructor: ~TrueFalse()**. Libera los recursos de un objeto de la clase TrueFalse.
- **Operador: TrueFalse& operator =(const TrueFalse& tf)**. Operador de asignación para la clase TrueFalse.
- **Consultora: string Categoria()**. Esta función es heredada de la clase Pregunta y nos devuelve la Categoría a la que corresponde la pregunta en forma de string.
- **Consultora: string Nombre()**. Esta función es heredada de la clase Pregunta y nos devuelve la Nombre asignado a la pregunta en forma de string.
- **Consultora: string pregunta()**. Devuelve el texto de la pregunta en forma de string.
- **Consultora: string correcta()**. Devuelve el texto de la respuesta correcta en forma de string.

D.1.5. Clase Numerica

Presentamos esta clase que es la que recoge las preguntas de tipo númericas del formato GIFT.

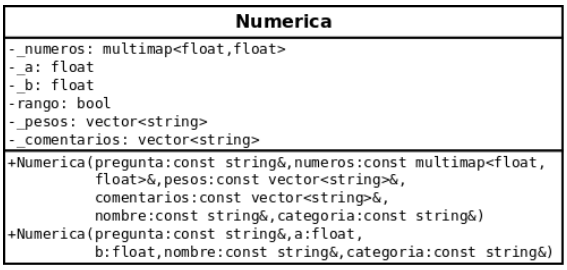

Figura D.6: Clase Numerica

Basicamente consta:

- **Constructor: Numerica(const string& pregunta, const multimap<float,float>& numeros, const vector<string>& pesos, const vector<string>& comentarios, const string& nombre, const string& categoria)**. Construye un objeto de la clase Numerica para la pregunta de tipo númerica por respuesta múltiple. Donde *pregunta* es el texto de la pregunta, *numeros* es un conjunto de pares de números, que indican la respuesta, *pesos* es un vector con los pesos para cada respuesta, *comentarios* es un vector con los comentarios a cada respuesta, *nombre* es el nombre que se le asigna a la pregunta y *categoria* es la categoría a la que pertenece la pregunta.
- **Constructor: Numerica(const string& pregunta, float a, float b, const string& nombre, const string& categoria)**. Construye un objeto de la clase Numerica para la pregunta de tipo intervalo. Donde *pregunta* es el texto de la pregunta, *a* es el comienzo del intervalo, *b* es el final del intervalo, *nombre* es el nombre que se le asigna a la pregunta y *categoria* es la categoría a la que pertenece la pregunta.
- **Constructor: Numerica(const Numerica& n)**. Constructor de copia de la clase Numerica.
- **Destructor: ~Numerica()**. Libera los recursos de un objeto de la clase Numerica.
- **Operador: Numerica& operator =(const Numerica& n)**. Operador de asignación para la clase Numerica.
- **Consultora: string Categoria()**. Esta función es heredada de la clase Pregunta y nos devuelve la Categoría a la que corresponde la pregunta en forma de string.
- **Consultora: string Nombre()**. Esta función es heredada de la clase Pregunta y nos devuelve la Nombre asignado a la pregunta en forma de string.

D.1.6. Clase Relacion

Presentamos esta clase que es la que recoge las preguntas de tipo relación del formato GIFT.

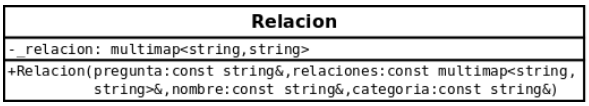

Figura D.7: Clase Relacion

Basicamente consta de las siguientes funciones miembro:

- **Constructor: Relacion(const string& pregunta, const multimap<string,string>& relaciones, const string& nombre, const string& categoria)**. Construye un objeto de la clase Relacion. Donde *pregunta* es el texto de la pregunta, *relaciones* es un conjunto con pares de relaciones, *nombre* es el nombre que se le asigna a la pregunta y *categoria* es la categoría a la que pertenece la pregunta.
- **Constructor: Relacion(const Relacion& r)**. Constructor de copia de la clase Relacion.
- **Destructor: ~Relacion()** Libera los recursos de un objeto de la clase Relacion.
- **Operador: Relacion& operator =(const Relacion& n)**. Operador de asignación para la clase Relacion.

- **Consultora: string Categoria()**. Esta función es heredada de la clase Pregunta y nos devuelve la Categoría a la que corresponde la pregunta en forma de string.

- **Consultora: string Nombre()**. Esta función es heredada de la clase Pregunta y nos devuelve la Nombre asignado a la pregunta en forma de string.

D.2. Clase Lote

Esta clase va a ser el contenedor de todas las preguntas que estén en el fichero que se quiera convertir. Mediante su constructor se lee y se transfieren las preguntas de memoria secundaria a memoria principal.

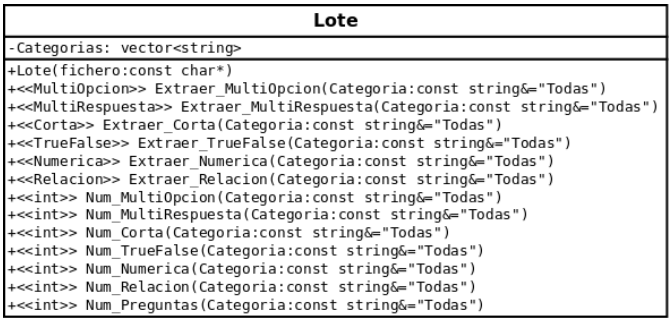

Figura D.8: Clase Lote

Consta de las siguientes funciones miembro:

- **Constructor: Lote(Fichero)**. Construye un objeto de la clase Lote, introduciendo las preguntas que estén en *Fichero*.

- **Destructor: ~Lote()** Libera los recursos de un objeto de la clase Lote.

- **Funcion: MultiOpcion Extraer_MultiOpcion(string Categoria = "Todas")**. Esta función devuelve una pregunta de tipo MultiOpcion de la categoría *Categoria* o por defecto de cualquier categoría.

- **Consultora: int Num_MultiOpcion(string cat = "Todas")**. Esta función devuelve el número de preguntas que hay del tipo MultiOpcion y de la categoría *cat* o por defecto de todas las categorías.

- **Funcion: MultiRespuesta Extraer_MultiRespuesta(string Categoria = "Todas")**. Esta función devuelve una pregunta de tipo MultiRespuesta de la categoría *Categoria* o por defecto de cualquier categoría.

- **Consultora: int Num_MultiRespuesta(string cat = "Todas")**. Esta función devuelve el número de preguntas que hay del tipo MultiRespuesta y de la categoría *cat* o por defecto de todas las categorías.

- **Funcion: Corta Extraer_Corta(string Categoria = "Todas")**. Esta función devuelve una pregunta de tipo Corta de la categoría *Categoria* o por defecto de cualquier categoría.

- **Consultora: int Num_Corta(string cat = "Todas")**. Esta función devuelve el número de preguntas que hay del tipo Corta y de la categoría *cat* o por defecto de todas las categorías.

- **Funcion: TrueFalse Extraer_TrueFalse(string Categoria = "Todas")**. Esta función devuelve una pregunta de tipo TrueFalse de la categoría *Categoria* o por defecto de cualquier categoría.
- **Consultora: int Num_TrueFalse(string cat = "Todas")**. Esta función devuelve el número de preguntas que hay del tipo TrueFalse y de la categoría *cat* o por defecto de todas las categorías.
- **Funcion: Numerica Extraer_Numerica(string Categoria = "Todas")**. Esta función devuelve una pregunta de tipo Numerica de la categoría *Categoria* o por defecto de cualquier categoría.
- **Consultora: int Num_Numerica(string cat = "Todas")**. Esta función devuelve el número de preguntas que hay del tipo Numerica y de la categoría *cat* o por defecto de todas las categorías.
- **Funcion: Relacion Extraer_Relacion(string Categoria = "Todas")**. Esta función devuelve una pregunta de tipo Relacion de la categoría *Categoria* o por defecto de cualquier categoría.
- **Consultora: int Num_Relacion(string cat = "Todas")**. Esta función devuelve el número de preguntas que hay del tipo Relacion y de la categoría *cat* o por defecto de todas las categorías.
- **Consultora: void Todas_Categorias(vector<string>t)**. Devuelve en el vector *t* todas las categorías disponibles de preguntas.
- **Consultora: int Num_Preguntas(string cat = "Todas")**. Devuelve el número de preguntas que hay en el Lote de la categoría *cat* o por defecto de todas las categorías.

Basicamente, así está formada la biblioteca que se libera. Para más información consultar la documentación realizada mediante el programa *doxygen* en la carpeta */prerres/doc/* o los propios ficheros fuentes contenidos en la carpeta */prerres/src/bibliotecagift*.

Bibliografía

[1] David Levinson. Quizzer. http://nexus.umn.edu/Quizzer/Quizzer.html#Quizzes, Consultado: Mayo 2010.

[2] Bernardo Cascales Salinas. *LaTeX: una imprenta en sus manos, ISBN: 849238-9-5*. Pearson Educación, 2003.

[3] Casiano Rodriguez-Leon. Documentación script para formato gift. http://search.cpan.org/~casiano/Gift-0.6/script/gift, Consultado: Septiembre 2009.

[4] Simple directmedia layer. http://www.libsdl.org/, Consultado: Septiembre 2009.

[5] Glenford J Myers. *The art of software testing, ISBN: 0-471-46912-2*. Hoboken: John Wiley & Sons, 2º edition, 2004.

[6] Antonio García Alba. Tutorial wiki de libsdl para la programación de videojuegos. http://softwarelibre.uca.es/wikijuegos, Consultado: Abril 2010.

[7] Carlos Lopez. Taller de sinfyg. http://tallersynfig.wordpress.com/contenido/, Consultado: Octubre 2009.

[8] Diego Sánchez Díaz. forja: Prerres: Información del proyecto. https://forja.rediris.es/projects/prerres/, Consultado: Mayo 2010.

[9] Documentación moodle sobre gift. http://70.86.170.226/en/GIFT_format, Consultado: Enero 2010.

[10] Formato gift básico. http://moodle.biblioredes.cl/help.php?module=quiz&file=formatgift.html, Consultado: Septiembre 2009.

[11] Tabla ascii. http://www.ascii.cl/es/, Consultado: Enero 2010.

[12] Gerardo Aburruzaga García Francisco Palomo Lozano, Inmaculada Medina Bulo. *Fundamentos de C++ , ISBN: 84-9828-007-9*. Servicios de Publicaciones de la Universidad de Cádiz.

[13] Banco de imagenes wikimedia. http://commons.wikimedia.org, Consultado: Enero 2010.

[14] Antonio Gacía Serrano. *Programación de videojuegos con SDL para Windows y Linux, ISBN: 8495836084*. Centro Andaluz del Libro.

[15] cpluspluc.com. C++ reference. http://www.cplusplus.com/reference/.

[16] Ministerio de Educación. Banco de imagenes y sonidos. http://bancoimagenes.isftic.mepsyd.es/, Consultado: Septiembre 2009.

[17] Partners In Rhyme Inc. Banco de sonidos. http://www.partnersinrhyme.com/soundfx/warsounds.shtml, Consultado: Marzo 2010.

[18] 20minutos. Banco de preguntas. http://www.20minutos.es/portada-trivial/, Consultado: Abril 2010.

[19] Meristation. Revista de videojuegos meristation. http://www.meristation.com, Consultado: Enero 2010.

[20] Wikipedia. Información sobre buzz! http://es.wikipedia.org/wiki/Buzz!, Consultado: Enero 2010.

[21] Eric J. Braude. *Ingeniería de Software. Una perspectiva orientada a objetos, ISBN: 84-7897-575-6*. RA-MA Editorial.

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
The "**publisher**" means any person or entity that distributes copies of the Document to the public.
A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.
You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.
In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.
"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.